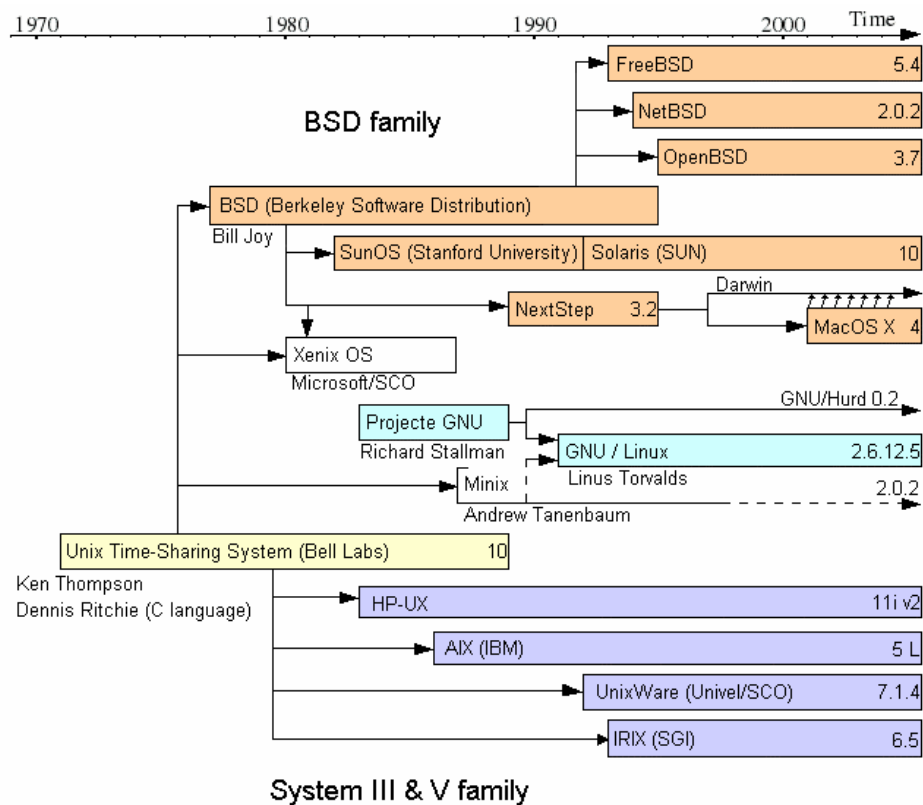


Managing Linux and UNIX Servers



Paul T. Ammann

Copyright

Copyright © 2006 by Paul T. Ammann

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. (<http://www.gnu.org/copyleft/fdl.html>)

Although every precaution has been taken in the preparation of this book, author assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein. (What do you expect? It's free.)

Trademarks

UNIX is a registered trademark of The Open Group. Linux is a registered trademark of Linus Torvalds. All other trademarks are the property of their respective owners. Throughout this book, trademarked names are used. Rather than put a trademark symbol in each occurrence of a trademark name, I state I am using the names on an editorial fashion and to the benefit of the trademark owner and with no intention of infringement on the trademark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

About the Author

Paul T. Ammann has written books for McGraw-Hill and Prentice Hall and has contributed 20 articles to MacTech magazine. He finds writing the bio the hardest part. He can be reached at pammann@gmail.com.

Art Cover

The art cover is from Wikipedia. Please see <http://en.wikipedia.org/wiki/UNIX>.

Table of Contents

Chapter 1: Introducing Best Practices	12
Systems Managers vs. Systems Administrators	13
The Need for Best Practices	13
Increased Security	13
Increased Reliability	14
Increased Cost-Effectiveness	14
Implementing Best Practices	15
A Holistic Approach to Best Practices	15
Infrastructure and Data Security	16
Backup and Restoration	16
Change Management	16
Performance Management	17
User Management	17
Fault Management	17
Task Automation	17
Defining Policy: The Crucial First Step	17
How to Organize a Policy	18
Service Level Agreements	19
Knowing UNIX: Overview and Idiosyncrasies	20
Linux and UNIX are Diverse	20
What's In a Name?	20
Unified Deployment	21
Unified Management	21
File-Centric Resource Access	21
Simplicity Found in Complexity	21
The Final UNIX Truth: Automation	22
Knowing Your Infrastructure	22
Infrastructure Servers	23
Data Servers	24
Application Servers	24

Interactive Servers.....	24
Workstations.....	25
Managing UNIX Management	25
Ticket Systems.....	25
Server and Application Documentation.....	27
Installation, Configuration, and Recovery	28
Service Layout	29
Network Layout	31
Conclusion.....	32
Chapter 2: Infrastructure and Data Security	33
The Security Policy.....	33
Policy Summary.....	34
Responsible Parties	34
Policy	35
Physical Security	36
System Security	36
Operating System Installation.....	37
Disable Incoming Network Access	37
Install Operating System from Vendor Media	37
Harden the Operating System	39
Disable Network Services by Default.....	41
Configure Better Logging.....	42
Update Operating System	43
Enable Incoming Network Access	43
Secure Applications	43
Application File Location	44
Access to Other Servers.....	44
Users and Trust	44
Passwords	44
Distributing Passwords	45
Managing Passwords.....	45
Vulnerability Analysis Tools	46
Intrusion Detection Systems.....	47

Kernel and Behavior Monitoring.....	47
File Integrity Checking.....	47
Network Security	48
Firewalls.....	48
NIDS	48
Insecure Communication Channels.....	48
Incident Response	49
Incident Identification	49
Investigation and Analysis.....	50
Containment and Remediation.....	50
Restoration.....	51
Documentation and Review	51
Disaster Recovery	51
Areas to Protect.....	52
Determine What Happened.....	52
Phases.....	52
Identify Disaster	52
Assemble Team.....	53
Recover Functions.....	53
Restore Full Service	53
Policy Compliance Monitoring.....	54
Computer Systems	54
Computer Usage	54
User Awareness and Training.....	54
Conclusion.....	55
Chapter 3: Backup and Restoration.....	56
Backup and Restore Policy.....	56
Identifying Needs.....	57
Define Restorations, Not Backups	58
Define Restoration Needs	58
Application Data	58
Applications and Operating System Files.....	59
Disk and Boot Volumes.....	60

Backup Server Software Data	61
Understanding Backup Technology	62
Types of Media	62
Magnetic Tape	62
CD-ROMs and DVDs.....	62
Disk-Based.....	63
Snapshots	63
Backup Drives	64
Single Tape.....	64
Autoloaders.....	64
Jukeboxes	65
Software	65
Protecting Your Backup Media	65
Organizing Media	65
Label Media	66
Saving Media from Destruction.....	67
The Art of Scheduling.....	68
Scheduling Backups.....	68
Scheduling Restores.....	69
Decentralized vs. Centralized Backups	70
Backup Automation: The Key to Survival	72
Custom Automation Solutions.....	72
Open-Source and Commercial Solutions.....	75
Monitor Problems Instead of Successes	75
Understanding Database Backups.....	75
Performing the Backup.....	76
Agent-Based Backups.....	76
Stopping the Database	76
Taking Snapshots or Mirrored Backups.....	76
Testing Your Procedures.....	77
How to Test	77
Operating Systems	78
Conclusion.....	78

Chapter 4: Change Management.....	79
Change Management Philosophy.....	79
Change Management Goals.....	80
Configurations	81
Software	81
Auditing.....	82
Reasons for Change Management.....	82
Better Use of Staff	82
Risks and Rewards.....	83
Increased Security	83
Increased Uptime	83
Improved Documentation	84
Reasons for Avoiding Change Management	84
Red Tape	84
The Status Quo	85
Define a Process	85
Refine the Process.....	86
User Base.....	87
Systems, Devices, and Networks	87
Implementation	88
Procedure.....	88
Step 1: Change Request	89
Step 2: Review and Approve.....	89
Step 3: Assign	90
Step 4: Research and Test.....	90
Step 5: Schedule and Execute	91
Step 6: Document.....	91
Step 7: Close Change Ticket.....	92
UNIX Change Management Tools.....	92
Request Tracking	92
Templates	92
Workflow Enforcement	93
Document Request Resolution	93

Ticket Systems	94
Change Implementation	94
Mixed Environments	94
Vendor Tools	94
Extensibility	95
Restoration	95
Auditing	95
Change Management Software	95
RCS and CVS	96
Cfengine	97
Conclusion	98
Chapter 5: Performance Management	99
Obtaining Performance Baselines	100
Performance Monitoring	100
Understanding the Numbers	101
Processors	101
Monitoring Tools	102
Multiprocessor Systems	104
Disk and File Systems	104
SCSI and IDE	105
Performance Considerations	106
Monitoring Tools	107
Real-World Performance Tuning	108
Memory	108
Real-World Performance Tuning	109
Network	110
Physical Network Performance	111
TCP/IP Performance	111
Monitor Protocols in Use	112
Focus on Important Applications	112
Bandwidth vs. Latency	112
Segment Your Networks	113
Load-Balance When Possible	113

Real-World Performance Tuning	114
Conclusion.....	116
Chapter 6: User Management	117
Account Management Elements	117
Account Information	118
Group Membership	118
Account Passwords	119
Home Directories.....	119
Centralized Management Using NIS.....	119
NIS Domain Names	120
Redundancy	121
NIS and Windows	121
Securing NIS	121
Restrict Network Access.....	122
Restrict Access to NIS Masters and Slaves.....	122
Patch Systems	122
Shadow Passwords and NIS.....	122
Centralized Management Using LDAP.....	123
The Directory Service	123
Location of Directories.....	124
Directory Management.....	124
Laying Out the Directory.....	125
LDAP, Linux, and UNIX	126
Distributed User Management.....	126
The User Management Policy	127
More About Identity and Access Management.....	128
Identity Management	129
Identity Integration.....	129
Conclusion.....	129
Chapter 7: Fault Management	130
Components.....	131
Disks	131
Disk Failure	132

Free Disk Space	133
Disk and File System Performance	134
Network Interfaces.....	134
Physical Layer.....	135
Data Link and Network Layer.....	136
Transport Layer.....	136
Application Layer.....	136
Applications	138
Configuration Management.....	138
Resource Usage	138
Fault Management	139
Centralized Monitoring.....	140
Conclusion.....	140
Chapter 8: Task Automation.....	141
Reasons to Automate	141
Eliminating Repetitive Tasks.....	141
Reducing Complexity.....	142
Documenting Tasks.....	142
When Not to Automate	143
Use One Scripting Language.....	143
Portability	143
Understandability.....	143
Power and Flexibility	144
Community Support.....	144
Network Applicability.....	144
Focus on Security	144
Network Security.....	145
Environmental Security.....	145
User Security	146
File Security.....	146
Umask.....	146
Temporary and Data Files.....	146
Don't Reinvent the Wheel.....	148

Design Scripts for Failure.....	148
Succeed Quietly, Fail Loudly	148
Log from Scripts.....	149
Informational Logs.....	149
Warning Logs	150
Error Logs	150
Keep it Simple	150
Use Abstraction	151
Centralize Scripts.....	152
Conclusion.....	152

1

Introducing Best Practices

Managing Linux and UNIX environments can be difficult. For administrators with years of UNIX experience, this statement might seem paradoxical because UNIX is an OS that is well suited to automation and large-scale deployments. But several factors contribute to both the perceived and real difficulties in using and managing UNIX. These factors include a complex administrative interface and the variations across systems that Linux and UNIX fragmentation (or perhaps more appropriately, divergence) causes.

These difficulties, combined with inadequate systems management training, can lead to a poor set of UNIX management procedures. The types of failure that improper management can cause include insecure installations, lack of availability in services, and unnecessarily complex and expensive management infrastructures. These failures affect end-user experiences and have an immediate and significant effect on UNIX systems' Return on Investment (ROI).

This book addresses Linux and UNIX systems management. That is, I focus on the how's and why's of managing a UNIX environment. This book will help you develop an understanding of existing systems management best practices and learn how to implement those practices in your environment.

The term *best practices*, as related to systems management, needs defining here. In most professions, including systems management, *principles* and *practices* determine how to best accomplish a goal. A *principle* is a basic truth or standard (e.g., a basic truth about how to manage Linux and UNIX systems). A *practice* is the way in which you implement a basic truth.

To better understand principles and practices, let's consider an example. A primary security principle is defense-in-depth. This concept states that you need multiple layers of defense between an attacker and a target. These layers might include a series of firewalls and hardened OSs and applications. The security manager's solution (i.e., firewalls and hardened systems) is the practice that implements the defense-in-depth principle.

In terms of Linux and UNIX best practices, we aren't concerned with just any practices. We're interested in the set of practices that have proven to be the best way to implement the principles that govern Linux and UNIX systems management.

Although principles rarely change, practices often change over time. This book discusses the current best practices for Linux and UNIX systems management. I cover best practices that implement several core Linux and UNIX management principles.

In this chapter I discuss several topics that are important to your understanding of how to apply Linux and UNIX best practices. For clarity throughout the book, I typically use the term *UNIX* rather than *Linux and UNIX* to refer to all UNIX and UNIX-like systems.

Systems Managers vs. Systems Administrators

This book is targeted toward systems managers rather than systems administrators. A *systems administrator* performs the daily administration tasks for a UNIX system. These tasks can include reviewing log files, installing new servers, verifying backups, and creating user accounts. A *systems manager* oversees the design and implementation of a UNIX environment. Although duties can overlap between systems administrators and systems managers, the goals of the two roles are different. In short, a systems administrator is concerned with the daily management of a UNIX system, whereas a systems manager is concerned with long-term management. Although this book will provide systems administrators with a solid foundation in planning and executing both short- and long-term management tasks, systems managers will most benefit from the book.

The Need for Best Practices

Most companies have a set of internal procedures that the company has developed over several years. Best practices, especially as I discuss in this book, don't necessarily take precedence over a company's internally developed procedures. Instead, a set of best practices is an adjunct to a set of internally developed procedures. Best practices should provide a company with a more solid foundation on which to customize industry-wide practices to its needs.

Best practices often focus on higher-level issues than internally developed procedures do. Because of their community- and industry-wide development and testing, best practices offer companies more ideas and solutions than internal procedures offer.

Most people can verbalize the importance of following best practices but don't understand how doing so will affect their organization's performance. In regards to Linux and UNIX solutions, following best practices increases the security, reliability, and cost-effectiveness of those solutions, as I discuss in the following three sections. Moreover, security, reliability, and cost-effectiveness directly and immediately affect the services an organization provides.

Increased Security

In years past, organizations running UNIX have been attacked, either directly or indirectly, by various methods. These attacks range from deliberate Distributed Denial of Service (DDoS) attacks to internally mounted espionage attempts through hacking or exploitations of network file system weaknesses, such as the weaknesses in NFS.

Over time, entire sectors of our digital infrastructure, ranging from the military-industrial complex to higher education institutions, have adjusted their methods of managing UNIX

systems to reduce their exposure to these attacks. The solutions employed have either succeeded or failed. Solutions that fail are thereafter rejected, whereas successful solutions merge and further develop into best practice solutions for protecting UNIX systems against attacks. These methods have slowly been absorbed into UNIX systems management best practices not because of regulations or law but because doing so made sense.

No single best practice exists for security or any other area of systems management. Therefore, best practices typically include a range of solutions for a variety of problems within a particular area of systems management.

Increased Reliability

Much like security best practices have evolved over time, UNIX reliability best practices have developed through years of systems managers' and front-line systems administrators' experience. Organizations with large UNIX environments have tried many reliability solutions and have incorporated the most effective solutions into reliability best practices. Using those best practices lets you take advantage of others' work and experience.

An example of a best practice that affects a UNIX network's reliability is centralized configuration management. Before the widespread adoption of UNIX and other mini- and microcomputer OSs, most systems were based on mainframes. These systems offered just one point for services configuration because only one machine existed to manage. But as UNIX systems spread, managing multiple systems became more important.

Unfortunately, managing many systems becomes more complex as the number of systems increases. Systems administrators eventually pushed for the ability to manage system configurations from a central location. They wanted revision control to track changes and to push changes out to the systems they managed. This method reduced the number of systems a systems administrator needed to visit when implementing changes. In addition, systems administrators had a better way to back up and restore configurations to systems. These changes translated into more reliable systems, because systems damaged by bad configurations were easier to restore and systems administrators made fewer mistakes.

Increased Cost-Effectiveness

The most important contribution you make to your company is its profitability. If the value of the services that your UNIX environment provides exceeds the cost of providing those services, then you are positively affecting your company's position in the global marketplace.

One of the most logical reasons for using best practices is that doing so saves money. Best practices have been developed externally, field-tested for you, and approved by a large portion of the UNIX community. Following best practices reduces or eliminates your need to extensively test most management methods. Instead, you can focus on previously proven solutions.

Implementing Best Practices

When a company encounters a set of best practices that will alter systems management, the staff often is reluctant to change. This desire to maintain the status quo is natural. When considering best practices, you must examine on a case-by-case basis which solution works best for your environment: best practices or internally developed procedures. Best practices usually prevail because they are widely accepted by others in the industry and because companies feel pressure to follow industry standards. However, following internal procedures might work best in some cases.

If you determine that following best practices is a better long-term solution, you need to create a program for a well-documented switch from your company's internally developed procedures to the methods that best practices define. In addition, you should convert to the new methods slowly rather than make sudden and radical changes. Methodically switching from one set of procedures to another lets you easily monitor the results of your changes and quickly abort changes that produce negative effects.

An important best practice to follow is to document on paper the procedures your company follows for systems management (and indeed for any task necessary to the company's operation). A useful test to determine which information to document is to imagine what would happen if a key employee vanished from the company.

A Holistic Approach to Best Practices

Managing UNIX is complex, and no single book can comprehensively cover the topic. Instead, I focus on core issues that face every UNIX systems manager. Figure 1 shows the seven UNIX management topics that I discuss and how they relate to best practices for managing your systems and servers. I discuss these seven management topics in Chapters 2 through 8.

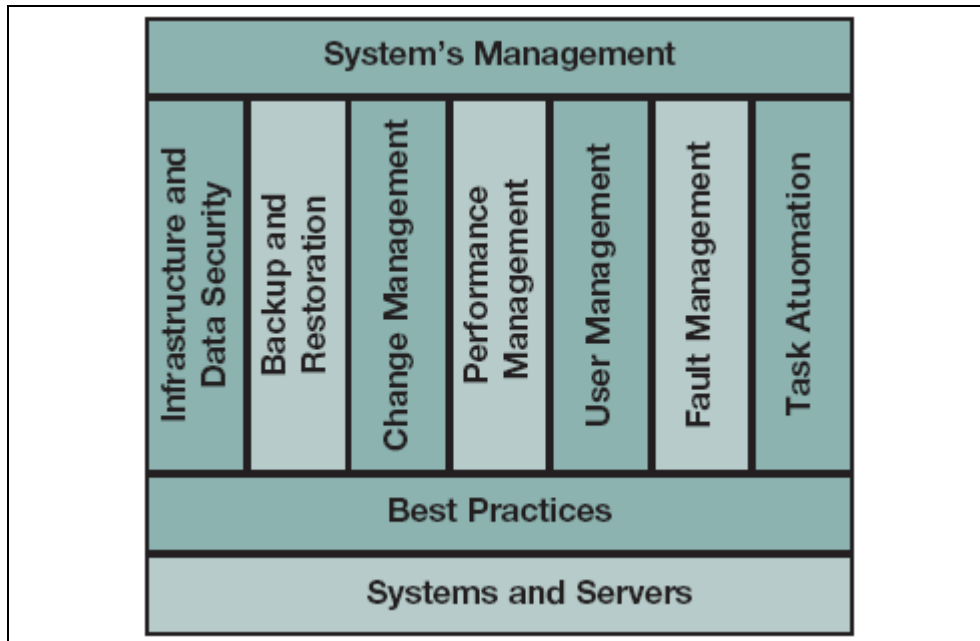


Figure 1-1. Areas of Focus

Infrastructure and Data Security

Despite what IT staff might say, IT departments often overlook security. The reason for this oversight isn't always funding but can be a problem of priorities. Because support staff is under constant pressure to maintain and create new and innovative services, they sometimes don't seriously consider how to securely build and maintain those services until the last minute—if at all. In Chapter 2 I concentrate on how to design, build, and maintain security in modern UNIX networks' physical and electronic realms, with an eye toward disaster recovery.

Backup and Restoration

Performing backups is a task that systems managers love to hate. Deploying a comprehensive backup procedure can be expensive, time-consuming, and labor intensive. Yet a good backup procedure is a vital component of a management plan. In Chapter 3 you will learn how to best design, implement, and verify reliable backups for UNIX systems.

Change Management

One of the most difficult aspects of systems management is managing change. The core concept of change management is to have the ability to properly schedule, execute, and verify changes to systems. Chapter 4 focuses on how to perform the actual scheduling, execution, and verification of changes in your UNIX environment and how to minimize the disruption that those changes cause your end-users.

Performance Management

Most environments don't have a well-defined and executed performance management policy and procedure in place. This lapse is surprising, considering the true cost of a poorly performing system. Chapter 5 discusses the performance information you need to monitor in a UNIX system, as well as how to utilize the information you gather to tune your environment and maximize performance.

User Management

One of the most complicated issues that organizations face is user management. User management includes not only creating and deleting accounts but also monitoring user activity and assigning roles and rights, among other concerns. Chapter 6 discusses these issues.

Fault Management

Fault management focuses on detecting errors in your UNIX environment, such as failing disks, resource overutilization, and key processes stopping (e.g., the Apache httpd Web server daemon failing on a Web server). The information in Chapter 7 will help you quickly and efficiently respond to faults and exceptions in your UNIX networks.

Task Automation

Task automation is the process of developing tools to handle repetitive tasks (e.g., restarting a stalled process, distributing account information about a new user to your servers). To accomplish task automation, you can use scripting tools such as the Bourne shell and Perl, as well as higher-level management tools from companies such as NetIQ and Hewlett-Packard (HP). Task automation is nearly synonymous with UNIX because UNIX provides a vast toolset for systems administrators to perform daily maintenance and systems managers to provide a more comprehensive management infrastructure. Chapter 8 integrates ideas from Chapters 2 through 7 into a more comprehensive set of policies and procedures for automating tasks in UNIX.

Defining Policy: The Crucial First Step

Policies should determine how your company uses computing resources. Resource use includes everything from end-user use to change management. Policy dictates procedures and guidelines, which in turn implement the policies' directives. To adequately discuss best practices in UNIX environments, we must begin with policy development.

Depending on legal and regulatory needs, a policy might be focused and straightforward or elaborate and often difficult to implement. The policies I discuss are specific to the needs of a UNIX systems manager who focuses on properly managing his or her UNIX environment to ensure high uptimes and service availability; these policies don't take into consideration a specific industry's legal requirements. You should be able to deploy the policies from this book in your organization. The policies that I define here will drive all the procedures that I discuss later in the book. Your company should follow the same rule: Procedures should be derived from policies.

How to Organize a Policy

Policies range from simple and effective to overly complex and ineffective. The policies in this book are straightforward and follow a standard, consistent format. Figure 2 shows the policy creation cycle.

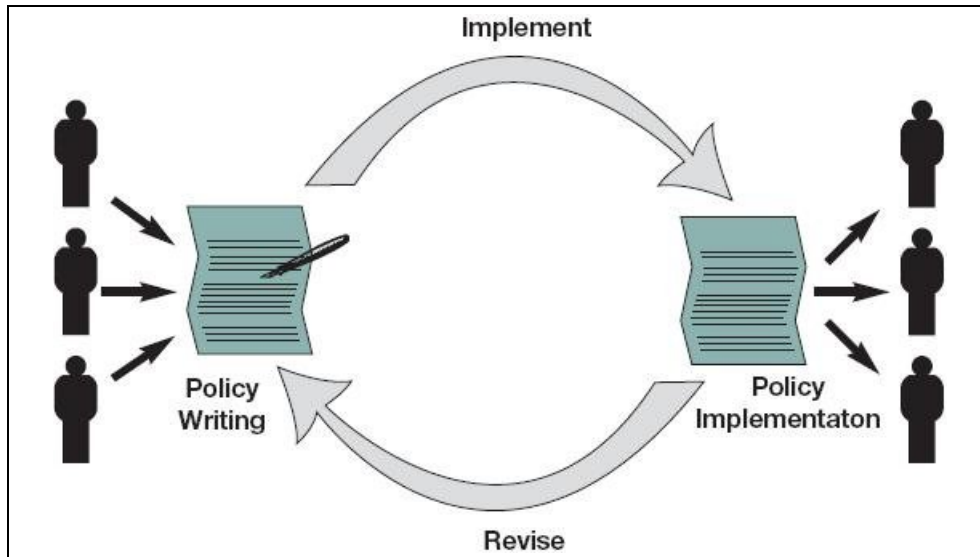


Figure 1-2. Policy Creation Cycle

Developing a usable policy typically involves a loop of two phases: policy writing and verification. In the policy writing stage you put pen to paper and work with all the involved departments to develop the overall policy structure and content. During verification the policy is put into place and field-tested. As you get feedback about the policy's real-world effects, you revise the policy. After you make revisions, you again put the policy into effect. This cycle continues for several iterations and can last from a few months to a few years, depending on your organization's size and how thoroughly a policy must be designed and written.

The need for this cycle of policy writing and revision is perhaps best illustrated with an example. I once worked as a UNIX systems manager for a company that developed proprietary software. This company needed a comprehensive security policy that controlled source code distribution.

After obtaining management approval, I started writing the policy. I worked with the software development team's head and other key company staff to develop the policy. When we implemented the policy and compared it to the company's daily activities, we quickly discovered that the policy didn't cover all software distribution methods (e.g., via email to an offsite programmer). We embarked on another round of policy writing and approval, then a new implementation. We finally achieved a policy that worked with how the company operated, rather than forcing the company to change its processes to follow a policy that didn't specifically address the company's day-to-day needs.

A policy has several important sections: Policy Summary, Responsible Parties, and Policy. The Policy Summary is simply a summary of the policy in question. This area should succinctly list the topics the policy covers. The Responsible Parties section lists

the people or roles that own and are authorized to enforce the policy. Finally, the Policy section contains the actual policy.

Service Level Agreements

Although they aren't directly related to best practices, service level agreements (SLAs) can be powerful tools in your collection. One way to define SLA is a policy between a resource provider and a resource consumer that determines an acceptable level of service. The SLA might contain a series of penalties against the service provider if the specified performance levels aren't met. SLAs define the minimum level of service that is acceptable to the consumer. For a UNIX network, such services might include email, Lightweight Directory Access Protocol (LDAP), or file storage.

Defining the service is typically easy. A more difficult task in establishing an SLA is defining the minimum level of service. For example, suppose that an SLA covers the service of access to an LDAP directory. The UNIX Services department centrally maintains this directory, which in turn powers several applications maintained across the organization. If access to the directory is impossible, obviously the SLA is violated. But what if users are able to connect, although a response takes several minutes? Because of this type of scenario, most SLAs define service levels in two ways: uptime and response time.

Uptime is the amount of time that a service is available and usable. Note that usable doesn't mean fast or responsive—simply that the service is up and available.

Most users care more about response time than uptime. Response time is the amount of time that a service takes to respond to a request. In a larger context, response time is what most people mean when they talk about end-to-end measurements, which are important in performance management.

Service load is how heavily a system is used. For example, service load can refer to how many services are running, as well as to the CPU, memory, and disk load.

As Figure 3 shows, sometimes no direct relationship exists between uptime and service load. However, a relationship always exists between response time and service load.

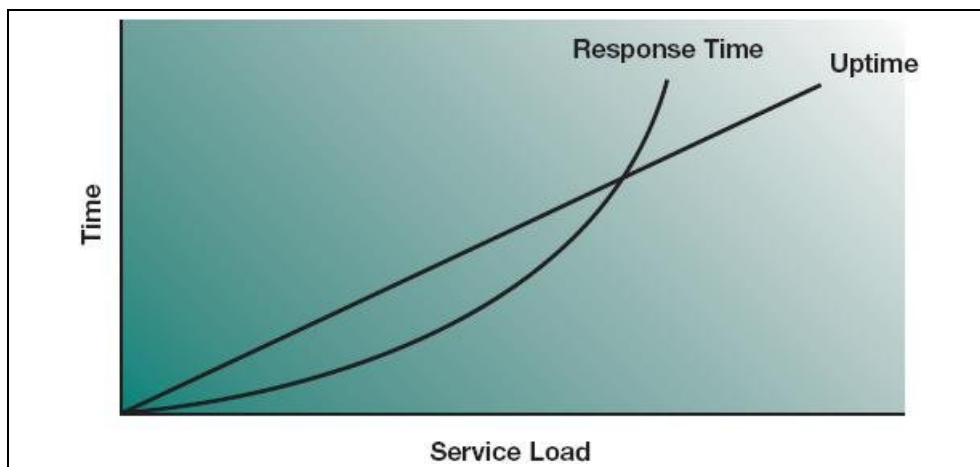


Figure 1-3. Service Load, Uptime, and Response Time

This book doesn't directly address SLAs. However, these agreements can form a powerful foundation for communication and even funding between your services group and other departments.

Knowing UNIX: Overview and Idiosyncrasies

Now that we've covered the policy side of systems management, let's consider some issues that affect UNIX manageability.

Linux and UNIX are Diverse

UNIX isn't an OS so much as it is a universe of OSs. Unlike most other systems, UNIX offers an incredible range of variation in implementations. As Figure 4 shows, all UNIX implementations trace their roots to the original UNIX developed at AT&T Bell Laboratories.

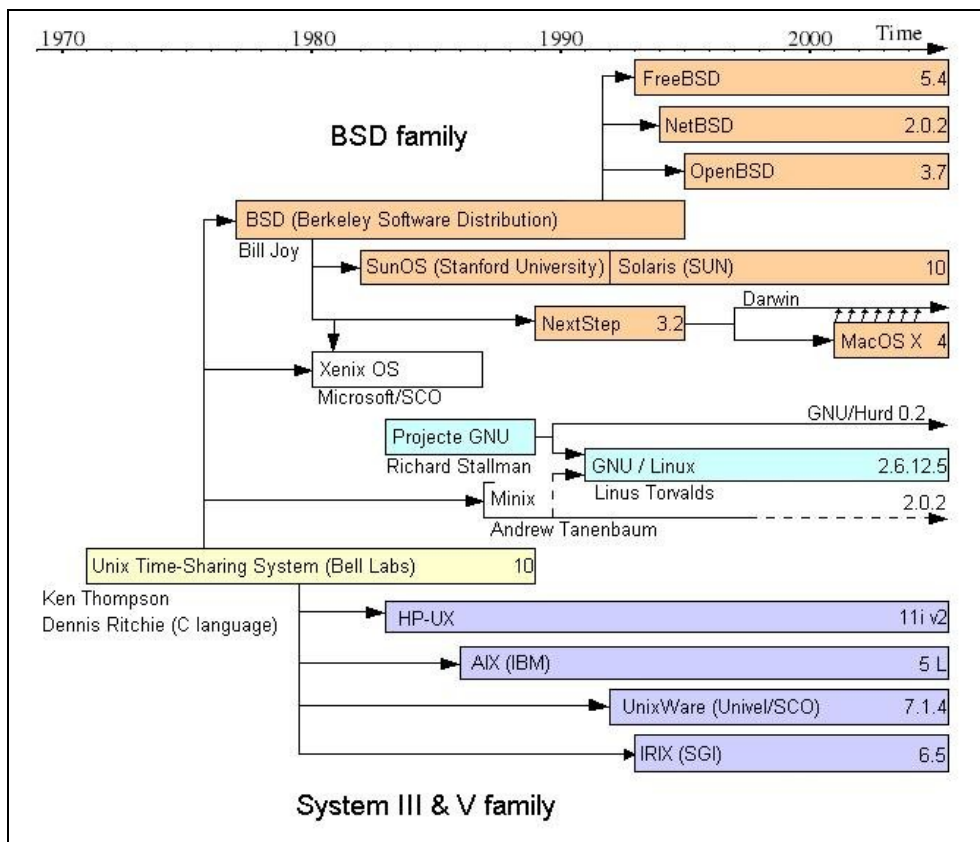


Figure 1-4. A Brief History of UNIX

What's In a Name?

Although most people consider UNIX to be a single OS or the entire set of UNIX and UNIX-like OSs, the word (or brand, as The Open Group notes) UNIX has a specific meaning. To receive the UNIX label, a vendor must pass The Open Group's

(<http://www.opengroup.org>) UNIX certification program. This certification requires that the OS support the Single UNIX Specification, pass a series of The Open Group's tests, and meet various other requirements. Obtaining the UNIX brand can be a long and expensive process. UNIX-like OSs such as Linux might meet most or all of the specifications necessary to be a UNIX-branded OS but still not *be* UNIX until the certification process for that specific Linux distribution is complete.

Over several years, the original UNIX split into two camps: BSD and System V. Alas, even this split wasn't clean. A wide assortment of UNIX systems branch off the two main branches, some merge back into a branch, and others (such as the UNIX-like Linux) live between the two branches—offering the best and sometimes the worst of both worlds.

Understanding UNIX's diverse nature is important in learning how to manage UNIX systems in an enterprise environment. Although diversity lets UNIX implementers focus their optimizations and development efforts on solutions for specific markets, diversity also increases the management load for organizations with multiple UNIX systems to support.

Several solutions exist for managing the problems that UNIX diversity creates. I discuss many of these solutions later in the book, but here I discuss two of the most popular approaches: unified deployment and unified management.

Unified Deployment

Organizations that follow the unified deployment tactic make concerted efforts to support only one flavor of UNIX. This solution is typically possible only with mainstream UNIX (e.g., Solaris, AIX) or a widely accepted distribution of Linux (e.g., Red Hat). Organizations that benefit from having various UNIX systems might not be able to take this approach, particularly if they tend to acquire or merge with other companies.

Unified Management

A more complex but often surprisingly more cost-effective solution is unified management. In this approach, one management infrastructure controls a company's various UNIX systems. Unified management, which is the tactic I use in this book, essentially homogenizes the management of a heterogeneous UNIX environment.

File-Centric Resource Access

A UNIX peculiarity is the almost obsessive focus on offering a file-centric interface to the OS. This focus is evident in several areas, from the heavy reliance on text-based configuration files to using device files in `/dev` to access devices. This file-centric view of the world has greatly contributed to UNIX's success. UNIX offers a cohesive interface because the OS typically provides developers with one core interface for most services and systems administrators with a consistent method of using text-based configuration files to configure services. This structure lowers development costs, lets developers and administrators leverage their knowledge across systems, and increases the portability of software across UNIX flavors.

Simplicity Found in Complexity

One of UNIX's strengths is the simplicity in the OS's apparent complexity. UNIX is a set of tools built on top of a kernel that gives systems managers a robust collection of simple tools to use together to create powerful results. In contrast, server OSs such as Windows

2000 typically offer larger and more complex management tools that address entire ranges of issues rather than having each tool focus on a specific task.

The Final UNIX Truth: Automation

As the previous sections imply, the key to understanding how to best manage UNIX systems is to understand automation. Other OSs, such as Windows 2000 Server, don't immediately offer the same high level of automation that UNIX has. Although new Windows system tools are constantly under development, until recently no Windows push existed for the same types of built-in, custom-developed, and commercial automation applications that are available for UNIX.

If you aren't automating your UNIX system's management, at least for mundane and routine tasks, then you're wasting resources performing work that the system should be doing. Examples of tasks you can automate are log filtering and alerts, resource usage alerts (i.e., CPU and file system overuse), and process monitoring. This book continually discusses ways to automate routine processes that can keep you updated on system status and help you build reports to enhance your organization's long-term growth.

Knowing Your Infrastructure

The UNIX systems you manage each perform some type of work. You can use the type of work that each machine performs to group the machines into classifications.

Classifying systems can be a difficult task. Many organizations manage multipurpose servers that perform multiple functions. Often, systems that offer firewall service for a remote location also offer other services such as email and a proxy or relay for authentication services.

Figure 5 shows the hierarchical classification system that I use throughout this book. A benefit of this type of system is that you can easily assign priorities for failure response time and general funding based on which level each classification is in. For example, a system in the Workstation layer will probably receive less attention if it goes down than a system in the Infrastructure layer would receive if it failed. Although this approach is common sense, making the classification concrete can be helpful in several instances, (e.g., budget discussions).

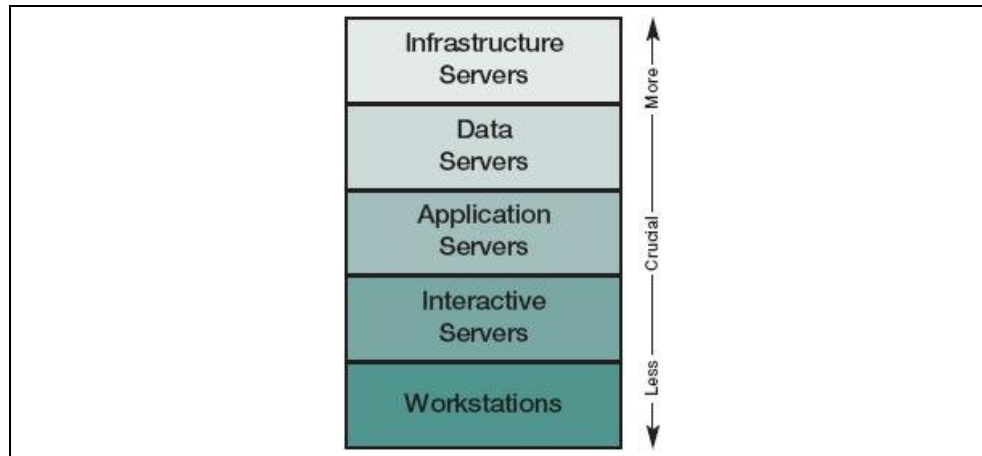


Figure 1-5. Hierarchical Role-based Classification System

Before I explain each classification, you need to know what kind of criteria determine a system's placement within the classifications. Classifying systems can be difficult when services are offered on multipurpose servers.

One classification approach is to identify the most important service a system offers, then place the system into the classification for that service. So, if a system offers IP routing between networks, the system would belong in the Infrastructure layer—regardless of which other services the system provided. This approach is simplistic and can lead to a misappropriation of resources because systems' importance is overemphasized or underemphasized. A solution to the problem of classifying systems is to move services for each classification layer to individual machines, or to group onto one machine the services that share a classification (providing for redundancy, of course).

Infrastructure Servers

An infrastructure server provides the services that comprise a UNIX environment's foundation. DNS is one of the most common services an infrastructure server provides. Any service that the network requires for proper operation is usually classified as an infrastructure service, and any server running such a service is an infrastructure server. A central concept behind the infrastructure classification is that if an infrastructure server fails, and no redundancy is built into the network, then several other services and possibly even network clients might fail.

Table 1 lists candidates for the infrastructure classification. As an example, consider the LDAP service. Many UNIX environments use LDAP as an authentication service; in these instances, authentication would fail without LDAP. LDAP is obviously an infrastructure service because it provides a core service that higher layers in the hierarchy require.

Table 1-1. Infrastructure Service Candidates

Service	Function
Administrative	Provides centralized servers for network management, server configuration, and forced failover.

Service	Function
DHCP	Provides IP address information to DHCP clients.
DNS	Provides domain name-to-IP and IP address-to-domain name mapping.
LDAP	Provides logon information for users and services.
Network Information Service (NIS)	Provides logon information for users and services and distribution of configuration maps.

Data Servers

Data servers usually fall into one of two categories: file servers and database servers. File servers provide network access to a file system, whereas database servers provide an abstracted interface to a set of data that the database server manages. In most cases, particularly with NFS in the UNIX world, file servers also provide an abstracted interface to the data in the file system—much like Server Message Block (SMB) and Common Internet File System (CIFS) provide users with a file system-independent method of accessing files on a Windows file server. (To be file system-independent means that the client doesn't need to understand NTFS to use Windows networking to access files on an NTFS file system over the network.)

Application Servers

Application servers are the computing world's middlemen. An application server is the server that provides the interface between a user and the back-end databases and processing servers. The interface is the method the user uses to get data into and out of the system. The application server also provides the business logic that determines what data is valid, how to format the data going into and out of the back-end databases and servers, and how to secure access to only authorized users.

The most obvious example of an application server is a Web server that an e-commerce Web site's customers access. In such a case, the Web server software Apache might provide the layer between users and the back-end databases. In addition, the Web server is where the Web site's e-commerce business logic runs, through a programming language such as PHP or ColdFusion.

Interactive Servers

An interactive server, or interactive logon server, provides users with remote logon capability. Telnet and Secure Shell (SSH) servers are interactive logon servers. Interactive logon servers let users access UNIX-based applications, check email, and use command-line tools such as Lynx (a text-based Web browser) to access the Internet. Servers that provide remote users with the graphical X Window System interface are also considered interactive servers because they give users and customers remote access.

Interactive servers have specialized needs. Because interactive servers give users local access to the file system and software, these servers have increased security vulnerability. In addition, interactive access servers often require specialized performance tuning. For non-interactive servers, tuning usually focuses on maximizing bandwidth and work done

per unit time. For interactive systems, the focus is usually on reducing latency—often at the expense of bandwidth usage.

Workstations

Workstations, or desktops, typically fall at the bottom of the classification system. A workstation lets a user run programs on the local CPU and memory; a workstation usually has a local file system. (An example of a workstation that doesn't have a local file system is an X terminal that has only enough resources to bring up an X server.) Depending on your company's organization, workstations can be cornerstone components that require more attention than the organization in Figure 5 implies.

Managing UNIX Management

One of the most important changes you can make when considering how to best manage your UNIX installation is to alter the way you manage your UNIX management. That is, you need to evaluate not only how you administer individual machines but also how you manage the entire network of UNIX systems.

If you approach systems management as you would approach systems administration, you run the risk of letting your staff devolve into merely a fire-fighting brigade. But if you view systems management with a holistic approach that focuses on problem avoidance rather than problem solving, you can fight the fires that break out, as well as adequately handle your business's long-term growth.

In general, long-term management solutions always include solutions to short-term needs. For example, addressing long-term backup, data integrity, and disaster-recovery needs also lets you quickly solve server failures.

Ticket Systems

One way to work toward problem avoidance rather than just problem solving is to track the problems your organization faces. Over time, you can analyze the data you gather and refine your management framework to better avoid the problems that continually occur. Using a ticket system helps achieve this goal.

A ticket system is a set of software that accepts a Help or change request from a user for review by a technician or manager. The ticket can contain just one problem or a long-term project synopsis. In either case, the ticket lets you track the problem and solution's progress as various employees and vendors work toward the solution. Figure 6 shows the flow of a ticket in a ticket-tracking system.

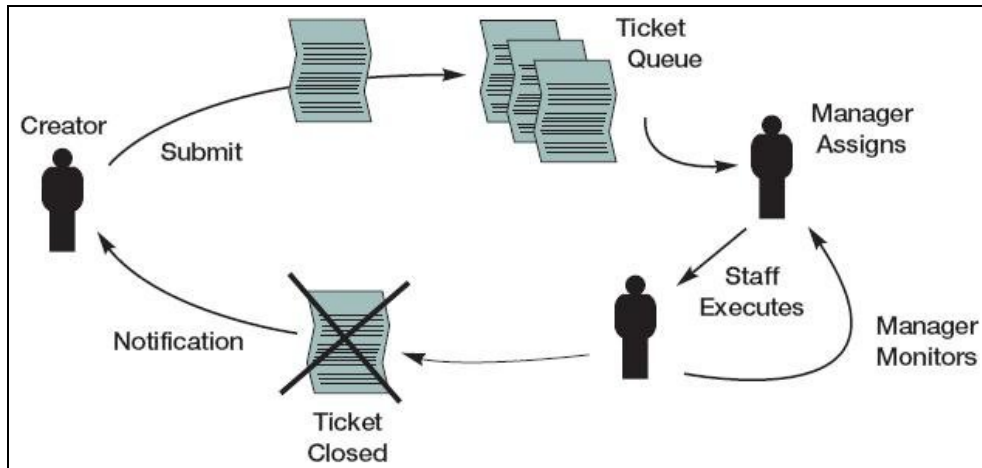


Figure 1-6. Ticket Flow in a Ticket-Tracking System

Ticketing systems vary and include commercial systems such as BMC Software's Remedy and open-source solutions such as Best Practical Solutions' Request Tracker (RT) and Open Ticket Request System (OTRS). Ticketing systems for smaller organizations also exist (e.g., PerlDesk). These systems aren't as full-featured as other solutions, but they are easy to deploy and use.

An important benefit of using a ticketing system is the ability to use a standard format to monitor the progress of multiple tasks and assignments from a centralized location. You can immediately determine task priorities and see who is working on each task and where each person is in the solution process. In addition, you can easily reassign tasks, change priorities, and alert users of their ticket status.

Figure 7 shows an open ticket in RT. Notice the amount of history available. Also note the ability of anyone working on the ticket to correspond directly with other ticket owners (i.e., those assigned to work on the ticket) and creators (i.e., whoever initially input the Help request).

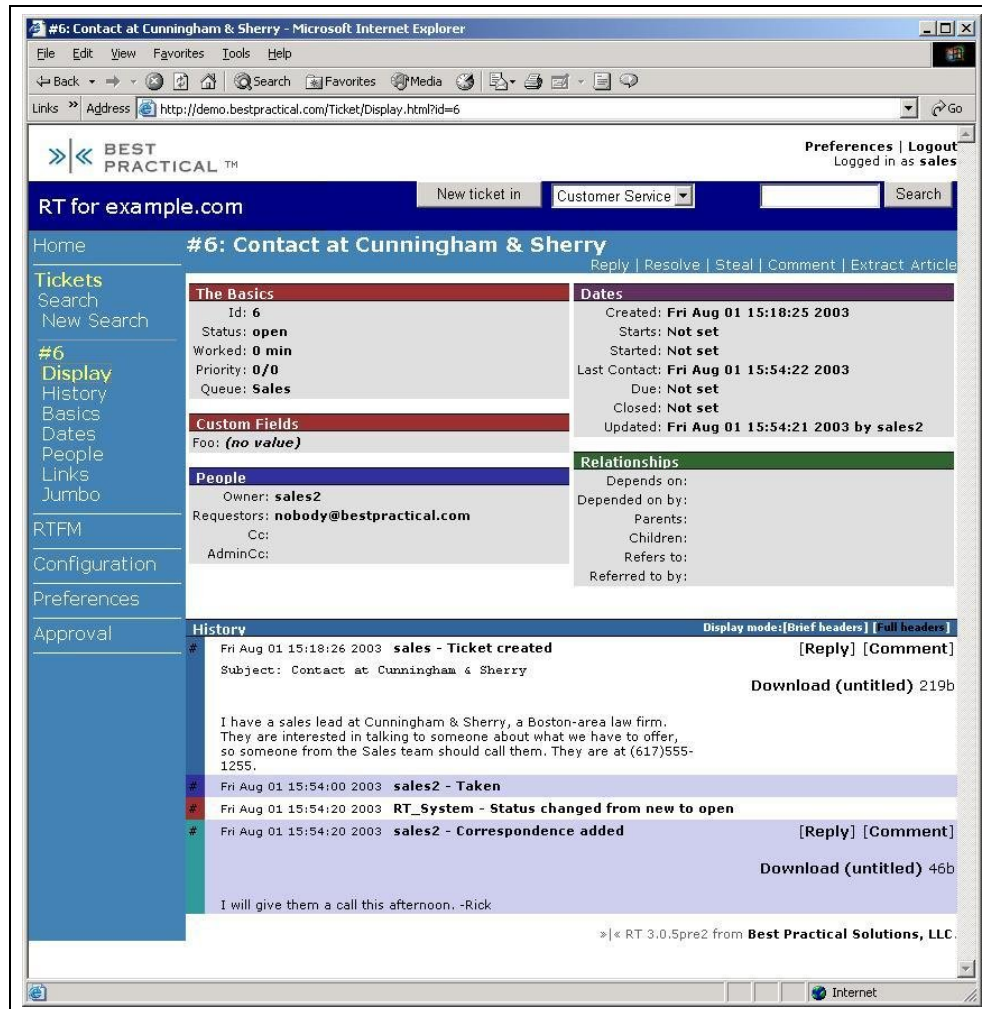


Figure 1-7. An Open Ticket in RT

Server and Application Documentation

Documentation is one of systems managers' and systems administrators' most neglected yet most important responsibilities. A systems manager must ensure that a methodical approach toward documentation is embedded in the management team's culture. Several forms of documentation must be maintained, including the three I discuss in this section: Installation, Configuration, and Recovery; Service Layout; and Network Layout.

When writing documentation, you need to consider your audience. If the document reader is a frontline technician (rather than a mid- or high-level administrator) who reinstalls remote office servers in remote locations, be sure to include sufficient details and background information.

You also must realize that documentation is often useful outside its original scope. For example, an auditor researching a system installation might decide to review the installation document to assess for variance between the system installation and the

documented installation. Your documentation needs to consider future readers, or at least direct these individuals to additional information that addresses their needs—perhaps information available in other documents.

Documentation must be accessible to the target community. If you place the documentation for a network installation on the network, accessing the information will be difficult or impossible if the network is down. When you build and implement a documentation system, be sure to include a procedure for offering multiple forms of access. Two common methods are to offer access over the network, in the form of a Web and FAQ site or a file server, and to place printed copies of each document in a well-indexed documentation file. You might want to create a core document set for IT access at your main site, and maintain essential documentation at each remote site. At each remote site you would then give at least one person, preferably a site manager, the role of documentation maintainer. The documentation maintainer works with the central site to ensure that the remote site's copy of the documentation remains current and accessible to anyone who needs it.

Finally, remember the most important reason and one of the most powerful incentives for creating and maintaining documentation: You can go on vacation!

Installation, Configuration, and Recovery

Installation, Configuration, and Recovery documentation concerns how to install and configure a new system with the designated services available for local or network use. Recovery is part of this type of documentation because the information is often necessary in crisis situations, in which you must quickly install and configure a new server because an existing server has failed catastrophically. Figure 8 is an example of this type of documentation.

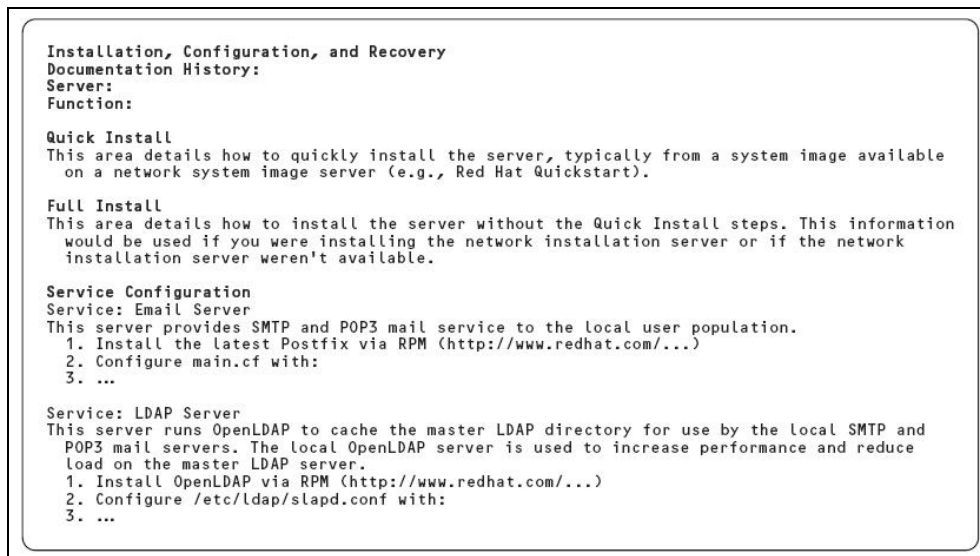


Figure 1-8. Example Installation, Configuration and Recovery Documentation

An alternative approach is to not include the detail service configuration section in each server's Installation, Configuration, and Recovery document but to instead maintain

unique service configuration documentation for each service and to reference that documentation in each server's Installation, Configuration, and Recovery document. A benefit to this approach is that you don't need to maintain several copies of the same information in multiple Installation, Configuration, and Recovery documents. A disadvantage is that you still need to write customized instructions for each server if your installations vary across machines. Figure 9 is an example of this alternative documentation.

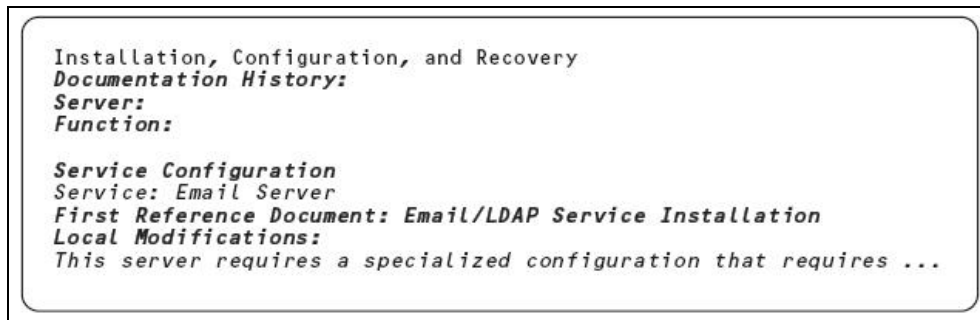


Figure 1-9. Example Alternative Installation, Configuration, and Recovery Documentation

Service Layout

Service Layout documentation is similar to Network Layout documentation but focuses exclusively on overall service use within a company. Systems managers use this type of documentation for tasks such as monitoring for required patches and determining which servers a possible service outage or upgrade will affect.

Unlike for Installation, Configuration, and Recovery documentation, a company typically has only a few Service Layout documents. Many organizations have only one master Service Layout document and one detail sheet that specifies service versions. Complex networks might require a master document and several detail layout documents for services that are related to one another (e.g., an e-commerce Service Layout document that describes Web, mail, DNS, and NFS services).

Figure 10 shows an example Service Layout diagram for a sales company's e-commerce division. Notice that the diagram doesn't include great detail. A systems manager would use this diagram to quickly identify areas of concern when making service changes and to determine which related services were affected.

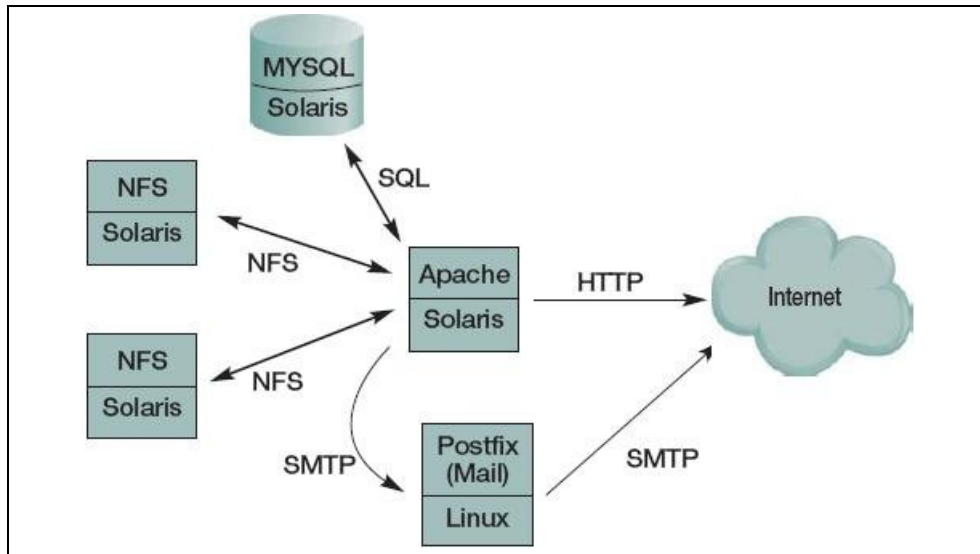


Figure 1-10. Example Service Layout

Table 2 shows the Service Layout detail document that would accompany the Service Layout diagram in Figure 10. This detail document identifies versions of service software, services' priorities, and services' responsible parties. The document lets you easily see who owns which service, which is necessary information in case you need to alert someone of a service outage. As an example, suppose you needed to immediately patch Apache because of a recently released exploit. You could consult Figure 10 to determine that no services rely on Apache, check Table 2 to determine who owns the service, and contact the E-commerce Division to schedule the upgrade.

Table 1-2. Table Caption Text Goes Here

Server	Service	Version	Priority	Purpose	Owner
db1.example.com	Database	mysql-server-323-34	3-Critical	Stores customer data and orders	E-commerce Division
smtp.example.com	SMTP	apache-1.3.29	2-Important	Relays mail from Web servers to company email server	E-commerce Division
www1.example.com	HTTP	apache-1.3.29	3-Critical	Application server for e-commerce Web site	E-commerce Division
www2.example.com	HTTP	apache-1.3.29	3-Critical	Application server for e-commerce Web site	E-commerce Division

The systems manager chooses the format for a detail document. I recommend spreadsheets because of their built-in sorting and filtering capabilities, although a large organization might require a dedicated database application.

Network Layout

Network Layout documentation is specific to network devices, physical and logical networks, and the integration of core services into the network. Like Installation, Configuration, and Recovery documentation and Service Layout documentation, Network Layout documentation is necessary in a well-documented and managed UNIX network.

Figure 11 shows a Network Layout diagram, which includes only the devices and networks in use on the network. This distinction separates Network Layout documents from Service Layout documents. A Service Layout document includes services such as DNS and DHCP, whereas a Network Layout document includes network devices and organization.

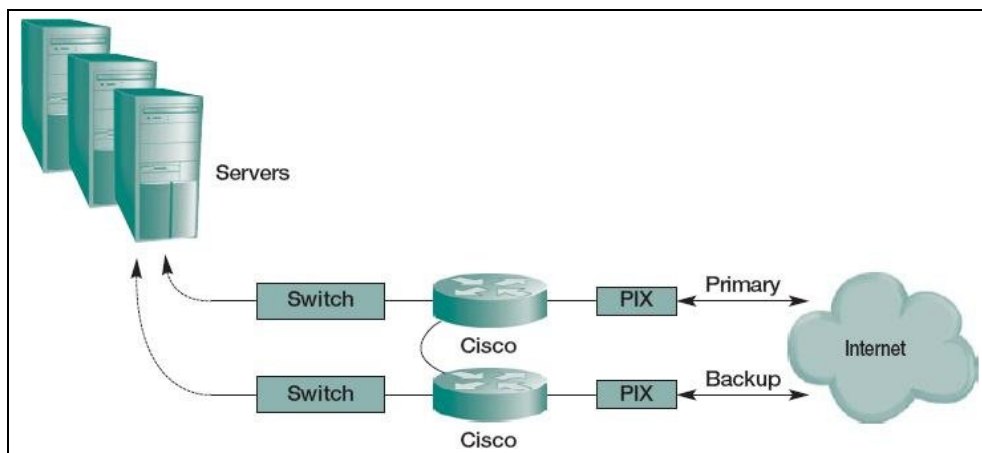


Figure 1-11. Example Network Layout

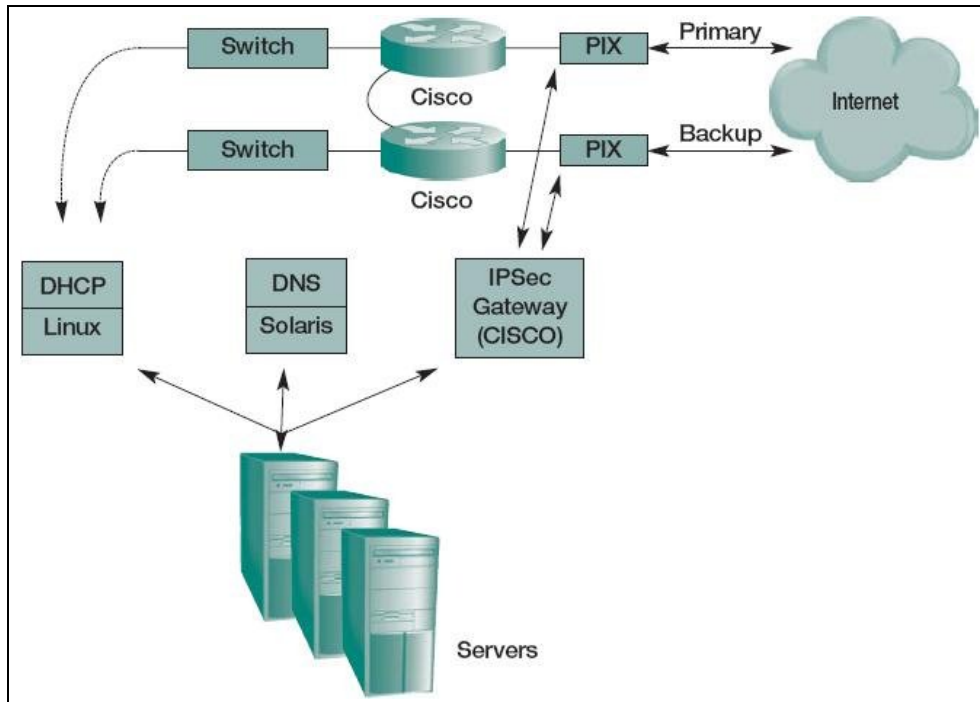


Figure 1-12. Modified Network Layout

Although the diagram in Figure 11 is concise, it isn't as cohesive as a document of this type should be. Figure 12 shows a modified diagram that includes core network services; this diagram provides a clearer picture of the services and equipment the network is using. However, a disadvantage of this presentation is that you either must maintain two identical copies of services, one for the Network Layout and one for the Service Layout, or you must reference the Network Layout when you install services during a new or recovered server installation.

Conclusion

This chapter covered a great deal of information. The topics I introduced here are merely a foundation for later chapters, which in turn provide even more in-depth information. All UNIX systems managers face similar problems in managing and maintaining their UNIX infrastructures. This book will help you discover the best practices and related software to ease your management burden and increase your environment's flexibility and resiliency.

2

Infrastructure and Data Security

Information systems security is important to businesses and governments around the world. Small businesses, corporations, and governments store crucial data on systems embedded deep within networks, as well as on the Internet's front lines such as on Web servers and email systems. UNIX systems especially have a long history of serving essential roles. Linux and UNIX systems provide infrastructure-level services (e.g., DNS, routing) and play an important role in e-commerce.

Several core security principles exist for securing servers, including least privilege, deny-by-default, and security-in-depth. In UNIX security, you can use a series of tested and proven practices, which I discuss in this chapter, to directly apply each of the principles. When relevant, I include information about how security affects business continuity and disaster recovery. Although I discuss disaster recovery separately at the end of the chapter, you need to integrate disaster recovery into all your security solutions.

The Security Policy

The first step in properly securing a UNIX installation is to develop a security policy that best addresses your organization's needs. Many companies hire consultants or rely on in-house expertise to develop a security policy. Companies that want to follow a standardized set of policy definitions can use free or commercial policy creation software packages to help develop a policy.

Organizations such as SANS (<http://www.sans.org>) and USENIX (<http://www.usenix.org>) offer free security policies. Most commercial software packages walk a security administrator through a series of questions, then create a policy's first draft based on the answers. The software creates a template that the security administrator can use to better tune the policy to the organization's needs.

In this chapter, I focus on a high-level security policy that encompasses a UNIX environment's core needs. Although I don't delve deeply into such a policy's details, the document I use as an example provides a good template for a real-world security policy.

Policy Summary

The policy summary includes the security policy's statement of purpose and the scope to which the policy applies. In my example, which Figure 1 shows, the security policy's purpose is to enforce a predefined set of behavior for systems managers when administering a UNIX server. This policy summary could also include end user requirements and requirements for network connectivity and security. However, your policies should be specific for your infrastructure's key elements.

The policy's scope typically specifies the systems, devices, or information that the policy covers. In my example, the policy's scope includes all the UNIX servers and workstations that the IT department manages.

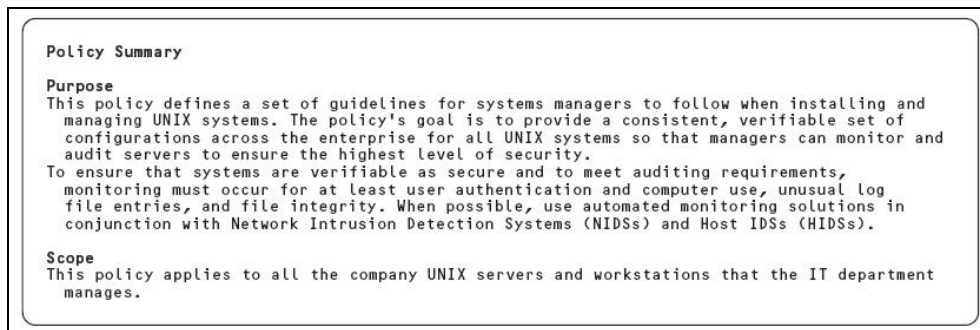


Figure 2-13. Example Policy Summary

Responsible Parties

The next section in the policy defines who is responsible for policy enforcement. Charging a team or group with a responsibility without giving them the required level of authority is a certain path to failure. Thus, the policy's Responsible Parties section must define a party that has both the responsibility and the authority to take action. Upper management must give the responsible party the right to directly punish or refer for punishment anyone who violates the policy.

Note: I once worked for a company that asked its systems administrators to write an information security policy. After we wrote the policy and worked with management to make revisions, we came to an impasse because management wanted administrators to be responsible for enforcing the policy but wouldn't give the administrators the authority to chastise or put on report those who failed to comply with the policy. Thus, administrators had no power over policy violators.

In my example, which Figure 2 shows, the responsible party is the Operating Systems group. This group is working to implement and monitor the security that the policy requires. In some cases, enforcement responsibility might fall to a dedicated information systems security or auditors team.

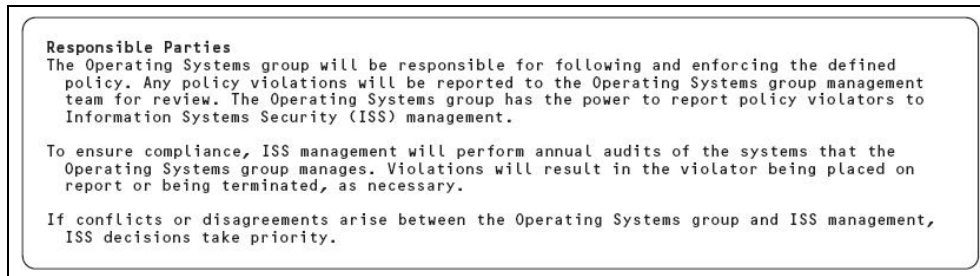


Figure 2-14. Example Responsible Parties

Policy

The security policy's most important element is the actual set of policies or guidelines to be followed. In many organizations, this section contains highly detailed do's and don'ts. In my example, which Figure 3 shows, I keep the policy section only detailed enough to form an initial template so that we can discuss major problems yet remain concise.

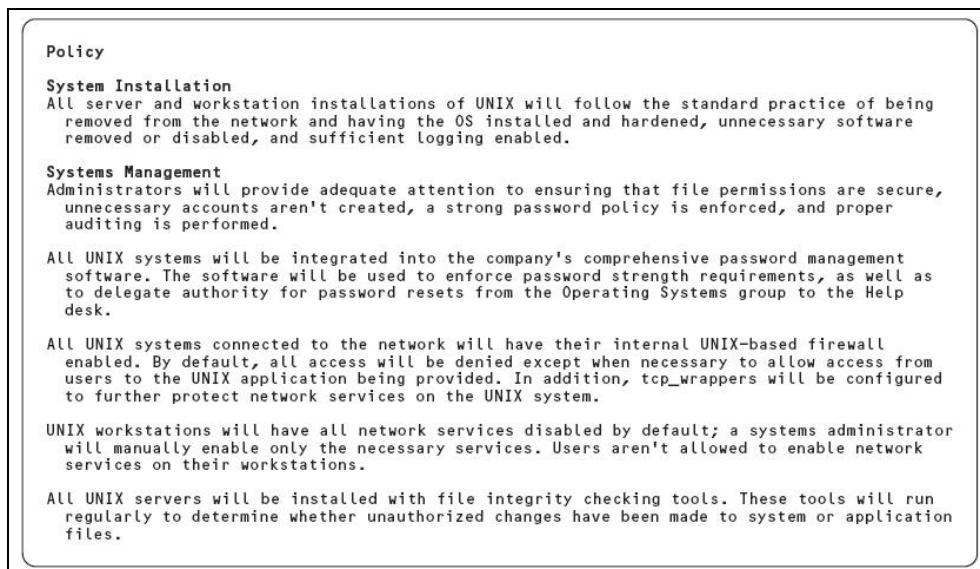


Figure 2-15. Example Policy

This policy provides a set of guidelines rather than specific requirements. In many situations, especially in large organizations, you might want to be specific (e.g., define the software to use to monitor logs and perform file integrity checking).

For a large collection of sample policies, see the SANS Security Policy Project (<http://www.sans.org/resources/policies>). For example, SANS has a password policy (http://www.sans.org/resources/policies/password_policy.pdf) that you could use to further strengthen my sample policy's password requirements.

Physical Security

Businesses often focus on the security of their UNIX OSs, applications, and data but neglect physical security. Physical security encompasses tangible items such as server and network hardware, server closets and rooms, cages and racks at collocation facilities, network cabling in conduits, and (most often ignored) storage areas for backup tapes. Ignoring physical security is dangerous because attackers are often insiders. In addition, even an outside attacker can often more easily walk into a company and gain unauthorized access to data or systems than attack the same company over the Internet or through dial-up access.

Physical security also plays an important role in disaster recovery. Disaster recovery involves quickly bringing back online key services to ensure business continuity. Physical security and disaster recovery go hand in hand because many physical controls can affect systems' survival during natural or man-made disasters. When planning your company's physical security, keep in mind events such as burglary, fire, tornado, hurricane, and loss of facility access.

To construct an effective physical security plan, define the systems you need to protect, the security perimeter around your systems, the threats you need to consider, and the possible defenses against those threats. As you consider your physical security needs, rate your systems by priority and direct the majority of your budget toward protecting your most important systems. For example, you need to keep network services, such as UNIX-based DNS servers, routers, firewalls, and database servers, under lock and key and in a location with fire suppressants. You can usually loosen your security requirements for less crucial systems, such as workstations (except in certain situations, such as on a military base). Although workstations are important to your network, you can easily replace workstations in case of theft or damage. Larger, more expensive server and network hardware are more difficult to replace.

For optimal physical security, you need to consult a physical security expert rather than rely on in-house expertise. Although systems managers and systems administrators are typically adept at securing UNIX systems, physical security requires a different set of requirements.

System Security

The most important aspect of security to UNIX systems managers is system security. Implementing system security in a UNIX environment is difficult for several reasons. First, many environments employ several UNIX flavors, ranging from Solaris and AIX to HP-UX. Each of these systems implements security differently. Fortunately, all UNIX systems share certain qualities (e.g., being file-centric). Thus, many differences in UNIX systems vanish when you view the larger picture of UNIX security, leaving the similarities in focus. However, you still need to focus on the differences as well as the similarities. For example, password management can require drastically different solutions as you cross between UNIX versions.

As I mentioned in Chapter 1, this book focuses on the unified management approach to managing UNIX systems. Unified management involves providing consistency across disparate UNIX systems rather than requiring just one UNIX flavor. In terms of security, unified management requires you to build a consistent set of procedures for tasks such as

securing servers during OS and application installation and building or purchasing a comprehensive solution for user and password management, monitoring logs and computer usage, and performing other management and security functions.

Operating System Installation

One of a systems administrator's core responsibilities is to install and configure UNIX systems. The systems manager must define and enforce a well-documented set of guidelines for systems administrators to follow when performing installations, because system security begins with a secure installation. In this section I focus on the most secure methods for installing UNIX. These methods vary slightly based on the version of UNIX you're supporting. However, the principles remain the same. To install UNIX, a systems administrator must disable incoming network access, install the OS from the vendor's media, harden the OS, disable network services, configure better logging, update the OS, and enable incoming network access.

Disable Incoming Network Access

Remotely attacking a system that isn't part of a network is quite difficult. Thus, you need to completely disconnect a system from the network before installing UNIX on the system. Doing so creates an *air gap*—the only thing connecting the UNIX system to the network is the air between them (i.e., no connection exists).

If you don't disconnect a new system when you install the OS, an attacker's machine or an already compromised system might detect the newly installed server (e.g., by using a ping sweep to locate new systems, then performing a port scan to detect network services that are enabled by default when the system is installed) and attempt to exploit the unpatched OS. If a system is compromised this early in an installation and you haven't installed file integrity tools, the only way you can determine that the system is no longer secure is to verify the installed files against the files on the vendor's installation media.

Note: Many people forget that modems are commonly found on servers. Some vendors ship servers with a getty process listening to the line, which creates a potential avenue for attack. When you unplug network cables, be sure to unplug all the cables—including telephone lines.

UNIX systems are often installed in a secured, mini-LAN that is dedicated to new installations. However, even these environments are vulnerable to attack. In fact, these machines often fall victim to attack faster than servers on the production network because many of the machines in these so-called secured LANs are left half-configured. If you decide to use a mini-LAN dedicated to new installations, be sure to completely restrict access to the Internet and the production network to outgoing access only—and even then only to servers or Internet sites necessary for updating newly installed UNIX systems. This action adds considerable security to the mini-LAN installation environment.

Install Operating System from Vendor Media

The next step is to actually install the OS from the vendor media. In most cases you'll use a copy of the media, which isn't necessarily a best practice but makes sense so that the original media isn't lost or stolen. You typically make a copy of the vendor media, then store the original in a safe or offsite location in case of a disaster.

The caveat is that using a copy of the vendor media exposes you to the risk that the copy has been tampered with. Although this risk is small in most organizations, it is still real. As you develop your procedures, keep in mind that most attacks are by insiders. One of the easiest ways for an insider to compromise a system is to do so before any security tools are installed and configured (i.e., when the OS is installed). Using verified media to perform installations minimizes this risk.

Note: Determining how to secure your media is an important concern. A good plan is to assign a manager to monitor the installation media and maintain a set of checksums. As Figure 4 shows, when the manager releases the media, he or she documents who receives the CDs or tapes. When the staff member returns the media, the manager can quickly verify that the checksum hasn't changed and thus verify that the media is still valid. Although this level of control might seem tedious, ensuring that your media is secure is important. Otherwise, later monitoring and integrity checking might be too late.

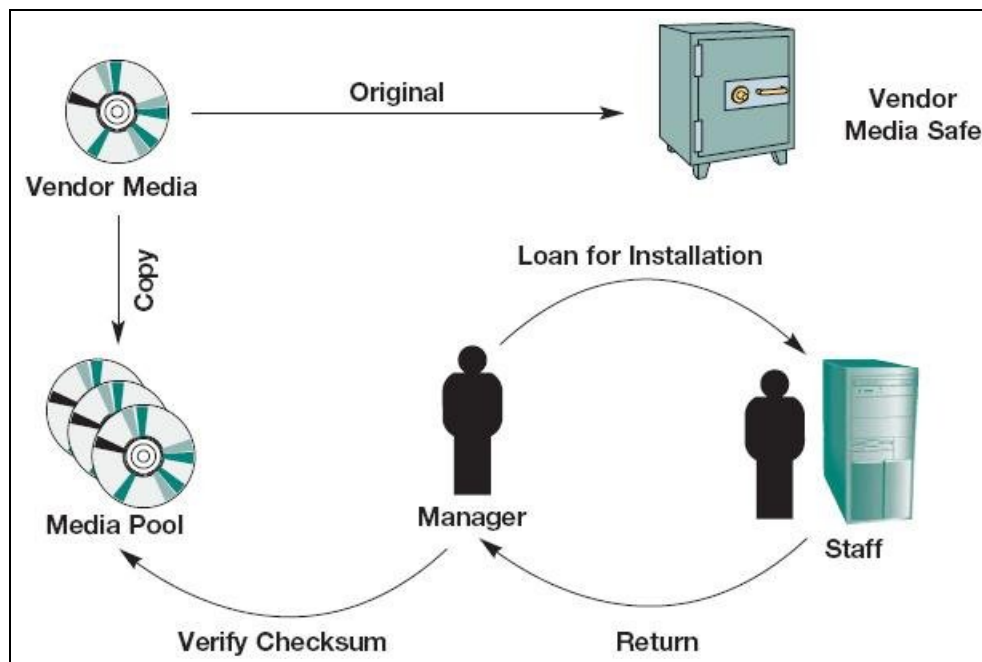


Figure 2-16. Media Verification

Another consideration when installing a new UNIX system is that vendors have a bad habit of installing too much software and enabling too many services by default. Increasing the amount of software and number of services a server provides greatly increases the server's vulnerability footprint. For example, most Linux distributions are well known for installing the Apache Web server by default even though attackers have significantly exploited Apache vulnerabilities in the past. Thus, you shouldn't install Apache unless necessary. Systems managers need to refrain from increasing servers' risk without justifiable cause. Vendors are slowly adopting a more minimalist stance than in the past and are including less software and disabling more services during a default installation. OpenBSD, an open-source BSD UNIX, takes this stance to an extreme and disables all services except Secure Shell (SSH).

Harden the Operating System

Hardening a server means performing a series of changes to a system to better secure the system. Hardening usually takes place during OS installation but also occurs as you develop new procedures, install new patches, and deploy new applications.

Note: You should always perform a new installation when you receive a new server or when a server with a preexisting OS comes under your management.

When hardening a server, you need to keep in mind the three principles I mentioned earlier: security-in-depth, least privilege, and deny-by-default. Security-in-depth means putting as many obstacles as possible between an attacker and potential targets. Least privilege means giving users only the level of access (i.e., privileges) they need to perform their jobs. Deny-by-default means that when you configure services, firewalls, or other security barriers, you should deny all incoming and outgoing services unless explicitly allowed.

Note: The alternative to deny-by-default is to allow all incoming and outgoing services by default unless explicitly denied. Although allow-by-default makes managing services easier than deny-by-default, allow-by-default makes maintaining security more difficult because you must constantly update a list of dangerous services, ports, and IP addresses to deny. Using deny-by-default is a best practice because a deny-by-default configuration's fail-safe mode is to deny service, thus protecting your UNIX systems in the event of a failure or a bad security barrier configuration (e.g., a firewall). Unfortunately, most UNIX systems installed with vendor defaults don't have the deny-by-default configuration. I discuss this problem throughout the chapter.

Secure User Accounts

Vendors often include unnecessary accounts on servers. These seldom-used accounts include FTP, uucp, guest, and news. You might encounter others as well, depending on your UNIX vendor. These unnecessary accounts exist because vendors often use default settings to build systems and therefore install unnecessary software. The easiest solution is to disable or delete the accounts.

Another concern when securing a system is knowing who has administrative account access. Special accounts, such as sys and root, have access to crucial OS files. The most powerful and therefore the most dangerous special account is root. Systems managers must define policies to control access to and monitor usage of the root account (or any account with a User ID—UID of 0).

You need to use multiple layers of security to control access to the root account. The first layer, and the most commonly used strategy, is to not give the root password to staff that don't need access to the account. In addition, you shouldn't allow root access directly from the network. Because most UNIX environments use SSH, compromising the root password over the network isn't the issue—but auditing root usage is difficult if administrators log on directly as root. You need to disable root logons in remote-access software (e.g., SSH) and require that administrators use the su command (or sudo, which I discuss) to become root.

If possible, you need to entirely disallow direct access to the root account. Use role-based access control (RBAC) if your OS supports it. Solaris supports a limited form of RBAC; you can delegate certain OS rights to specific users. Those users then have root-like privileges for the functions you specify. RBAC doesn't provide a comprehensive solution for delegating rights, because no general-purpose UNIX systems fully implement RBAC. Add-on solutions such as Symark's PowerBroker offer RBAC features for Linux and UNIX systems. Sudo is a widely used UNIX open-source rights-delegation tool. Sudo lets managers delegate rights in a more fine-grained fashion than giving everyone access to the root account. Sudo uses syslog to log every command that runs, as well as who ran the command. Sudo is an excellent tool for auditors; you should mandate its use for administrative functions in your environment.

Note: You need to keep the logs that RBAC and sudo generate longer than you keep regular server logs. You might not discover inappropriate privilege use for several weeks or months, so you need to be able to retrieve usage records for long time periods. If you don't keep these logs, you might have a difficult time finding credible evidence to prove privilege abuse.

Tighten File Permissions

As I mentioned in Chapter 1, UNIX is a file-centric OS. You use files to access everything from application configuration information to hardware devices. Because file permissions play an integral role in UNIX security, ensuring proper file permissions maintenance must be one of your security procedures' major focuses. You need to train your systems administrators to consider file permissions one of the most important elements of system security.

File and directory permissions modes can inform the OS of special modes to use when executing a file. An executable file can have two special modes: set group ID (SGID) and set user ID (SUID). With SGID the executable runs with the permissions of the group owner as opposed to the user running the program. With SUID the executable runs with the permissions of the file owner. Most host security analysis programs, which I discuss later, automatically search for and report SUID programs.

A common use of permissions in UNIX is to grant or deny read (r), write (w), or execute (x) access to a file or directory. UNIX has three groups of security permissions: user (u), group (g), and other (o). The user is the UNIX account that owns the file, group is the UNIX group that has group ownership of the file, and other is any user who isn't an owner or in the group that owns the file. In general, you should use the deny-by-default and least privilege principles to create files, which gives only the owner access. If necessary, you can also give the owning group access to key files. This strategy is useful when users share files. The other group should have access to only certain files—and in most cases (e.g., the /etc/passwd file) the access should be read only.

The umask system tool defines default security permissions for newly created files and directories. Settings are defined in a script that runs when the user logs on (e.g., in /etc/cshrc). Many remote file access tools, such as SSH File Transfer Protocol (SFTP), also need proper configuration to apply secure permissions when creating or copying files and directories.

Note: Ensuring that system and application files have the proper security permissions is on par with maintaining a strong password policy. Strong security permissions can protect you if a server account is breached, because most accounts wouldn't be able to make unauthorized changes to key files.

The way that UNIX traditionally implements file security is sufficient in most cases, but this method can present a challenge for administrators who want to implement fine-grained access control. Many UNIX systems, including Linux, now include ACLs as a complement to traditional file security. ACLs let administrators and users specify a list of users and their access rights, rather than restricting access control to the more limited user-group-other model. ACLs greatly simplify UNIX security administration.

Disable Network Services by Default

The best practice in terms of network services is to disable all services during system installation, then activate only the services you need. Decreasing the number of services a server offers lessens the server's vulnerability footprint, thus limiting possible avenues for attack. This principle is especially true for the services typically enabled on a default installation (e.g., FTP, Telnet), because the server software that offers these services has a history of exploits.

UNIX services usually start in one of two ways: from a startup script or through the Internet superdaemon inetd. To disable a service in inetd, remove or comment out the service line in `/etc/inetd.conf`. Following are examples of a service that is enabled and disabled.

```
#ftp  stream tcpnowait root    /usr/libexec/ftpd ftpd -l
ftp   stream tcp      nowait roott  /usr/libexec/ftpd      ftpd -l
```

Services commonly started from `/etc/inetd.conf` include Telnet, FTP, and Samba.

Inetd has a long UNIX history but is slowly being replaced by xinetd. Xinetd uses each service's configuration file to define the service rather than configuring all services in one central configuration file. For example, for xinetd the ftpd service would probably be defined in `/etc/xinetd.d/ftpd`, whereas in inetd the ftpd service would be defined in `/etc/inetd.conf`. To disable a xinetd service, change the configuration line *disable = no* to *disable = yes* in the appropriate `/etc/xinetd.d` configuration file. Xinetd also lets you specify which hosts can access the service, much like tcp_wrappers works. (I discuss tcp_wrappers later.) Xinetd gives you an additional tool in your security-in-depth arsenal.

The location of startup scripts depends on whether the UNIX you're using is based on BSD or System V (SysV). For BSD systems startup of most services begins with `/etc/rc`, whereas for SysV systems startup begins via a script in `/etc/rc.d/init.d`. In BSD you can comment out the relevant lines in `/etc/rc` to disable services, whereas in SysV you can use `chmod` to disable execute permission, as in the following example.

```
# chmod ugo-x /etc/rc.d/init.d/samba.sh
```

You need to restrict access to services that you want to leave enabled. Three methods let you restrict access to network services: firewall, application, and tcp_wrappers. With a firewall you can disable access at the TCP/IP protocol's packet layer. Tcp_wrappers, which I discuss below, provides a UNIX method of allowing or disallowing remote client access to local services. Finally, many network services let you specify which hosts have

access. Samba and Apache are example of these services; both let administrators specify which remote hosts can connect to the service.

Wietse Venema's `tcp_wrappers` protects TCP-based network services by letting administrators specify which remote hosts are granted or denied access to the network port the network service is using. `Tcp_wrappers`' most useful feature is that the service typically runs independently of the service being wrapped. Thus, the wrapped application doesn't need to know that `tcp_wrappers` is running it. Wrapping a service in `/etc/inetd.conf`'s configuration line lets you configure the service for `tcp_wrappers` protection. In addition, you can compile and link certain applications, such as the Apache Web server, with `tcp_wrappers` support. Although Web servers don't generally run from `inetd`, this feature lets them rely on one `tcp_wrappers` configuration for network access control.

Although I don't cover `tcp_wrappers` in depth, you need to know that its prevalence on UNIX systems makes it an excellent method for consistently protecting network services. With the proper configuration, you can enable logging of all network service access, such as SSH. This feature is especially beneficial when a server's logs are linked with a monitoring system or IDS.

Configure Better Logging

An attacker's dream is lax logging. An auditor's dream is strict logging. Of the two, satisfying the auditor makes the most sense. System logging in UNIX is primarily `syslog`'s realm. `Syslog` is typically implemented as a daemon that starts when the system boots up (i.e., `syslogd`).

If you're running multiple UNIX servers, you need to log to local log files and a central log host. To configure this logging in `/etc/syslogd.conf`, specifying that all log entries send to the remote host, as in the following example.

```
*      @logger.example.com
```

In this example, messages that send to the local `syslog` daemon also send to `logger.example.com`. You might want to restrict sent messages to only those related to security. However, a good practice is to log everything, then post-filter the results to drill down to the data you need. This method gives systems administrators access to all log entries on one central server, and systems manager and auditors have a complete history of the actions performed on managed servers. As I discussed earlier, any commands that `sudo` invokes log to `syslog`. Thus, using a central log server lets you build one monitoring application to monitor privileged access.

A central log server offers many additional capabilities. For example, a systems administrator or automated monitoring application can scan the collected logs to determine whether a pattern of access on several systems might indicate an attack. If you inspect logs only on a server-by-server basis, you might consider an anomaly an isolated incident rather than notice a pattern.

In observance of the deny-by-default principle, on most UNIX servers you need to disable the `syslog` daemon's ability to accept log entries over the network. For Linux's `syslogd`, you can add the `-s` option. To completely stop the local `syslogd` daemon from sending messages over the network, you can add `-ss`.

Note: A common error when first configuring a central logging host is to accidentally enable remote logging on the logging host itself. Doing

so causes the log host to send itself a log entry, which the log host sends itself again, and so on. Most syslog daemons aren't intelligent enough to stop this feedback, and the logs quickly fill up. Pay close attention when configuring these settings.

Just logging system events isn't enough. You must also monitor the logs and ensure that recorded events don't indicate noncompliance with a policy. At the end of this chapter, I discuss how to monitor for policy compliance and detect attacks and compromises.

Update Operating System

Vendors typically release new installation media only after several major updates to their OSs are released, so the OS installed from the media might be outdated. Therefore, the next step is to update the OS to the most recent patch level. (I discuss patch management in more detail in Chapter 4.)

Placing an unpatched server on the network is dangerous. Even if you disable all the network services, an unpatched kernel can give potential attackers a Denial of Service (DoS) opportunity. After you patch a server, you need to review the changes and possibly reapply the fixes you applied during the initial hardening. You need to repeat this cycle after each update: Update the server, then harden it. Systems managers benefit from automating all or most of the hardening process by using vendor-supplied tools or open-source or commercial solutions, or by developing in-house scripts to harden their specific versions of UNIX and the applications they provide.

Enable Incoming Network Access

After you've secured the server, you're ready to provide service to the network. You've installed the OS, hardened the server, and applied the most recent security patches. The final step is then to enable incoming network access.

Secure Applications

After you install a server, your focus changes from the UNIX OS to the target application. Applications are even more vulnerable to attack than are the underlying OSs. This security vulnerability exists because of improperly written UNIX software and inappropriate privilege use that violates the least privilege principle.

Note: Applications often run as root when they could easily run as a normal user. This scenario occurs because the application needs to listen on a privileged port (i.e., any network port between 1 and 1024), and the application must therefore start as root. However, after the application gains control of the port the application can drop its privileged status as root and change to another user.

You need to be careful when installing and configuring applications on your UNIX servers. Your installation procedure documentation must consider the following:

- Application program files', configuration files', and data files' locations
- Whether the application requires access to another server for operation
- The kinds of users the application will support and your level of trust in those users

I review each of these areas in more detail in the following sections. Your installation procedures need to address each area and provide guidelines for each supported application.

Application File Location

In general, UNIX applications follow one of two installation methods: Application files are in a directory specific to the application, or application files are spread across directories in /usr/local. A recent trend has been to install applications in /opt, but this approach is just a variation of the first method I mentioned. From a management standpoint, installing an application in its own directory is the better of the two options. This method lets you easily monitor for file changes (e.g., using file integrity monitoring software) and enable simple backups and restores. In addition, depending on how fast the application data files will grow, dedicating an entire file system to the application is a good idea—and having all the files in one location makes creating such a file system easy.

Access to Other Servers

Another consideration is whether the application requires access to other servers for operation. Many applications (e.g., Apache) require at least DNS access. In addition, many enterprise Apache applications are built using a scripting language embedded in the Web pages and require access to database services from Oracle, Sybase, or the open-source MySQL relational database management system (RDBMS). Therefore, as part of the application installation and configuration procedure, you must open access to these services to the Apache server. As always, you need to follow the least privilege principle and give Apache only the minimum level of access necessary for the Web application to function.

Users and Trust

Finally, you need to study the target user base for the application you're deploying, with a special emphasis on determining the danger of attack. For a Web application that is based on Apache and is available to the Internet community, the danger of attack is high. For a small, targeted application dedicated to a receiving department, the danger might be low. Although systems managers want to provide maximum security for all their applications, organizations have finite budgets allocated to security. Thus, you need to allocate the appropriate amount of resources for security depending on how crucial the target application is to your organization's operation and the danger that those with access to the application present.

Passwords

Users enter passwords to log on to servers and access applications. Most UNIX systems store account information in /etc/passwd and encrypted passwords and additional information in a shadow password file. When a user logs on to the server, the system scans the password database and verifies that the entered password matches the password in the database. This system is sufficient in many environments because it's simple to maintain if you're managing only one or two servers and because you can easily back up and restore the password and shadow password files. More complex environments require a more comprehensive system to distribute and manage passwords.

Distributing Passwords

Distributing account information, including passwords, has long been Network Information Service's (NIS's) realm. NIS is a set of protocols and software that Sun Microsystems developed to distribute UNIX configuration information in large networks. UNIX has slowly outgrown NIS, but NIS still exists in many environments. Unfortunately, NIS is insecure; you should use it only in a highly trusted network.

Lightweight Directory Access Protocol (LDAP) is a method of centralizing passwords that is finding its way into UNIX networks. LDAP over Secure Sockets Layer/Transport Layer Security (SSL/TLS) provides a secure protocol for verifying passwords. You can expand LDAP over SSL/TLS to include configuration information for everything from printers to applications. Use LDAP whenever possible.

How to support LDAP (or any authentication framework) varies across UNIX platforms. For systems that use the Pluggable Authentication Modules (PAM) interface (e.g., Solaris, Linux, FreeBSD), support can be as simple as configuring your servers to use LDAP PAM. For other UNIX systems, you might need to purchase commercial software or locate open-source solutions from your vendor or other users.

Note: Sun developed a new version of NIS known as NIS+. NIS+ addresses many of NIS's problems. However, NIS+ isn't as widely implemented and deployed as NIS. To use NIS+, you must restrict your supported UNIX systems to Solaris. (Linux technically supports NIS+, but the support is buggy and not actively developed.) Move to LDAP if possible because of its wide industry support.

Managing Passwords

Enforcing a solid password policy is important. Two main attacks against weak passwords are remote access attacks and password file attacks. In recent years, defense against these attacks has improved, in the form of temporary account lockouts for remote access attacks and shadow passwords for password file attacks.

Note: Surprisingly, for many years Linux systems offered shadow passwords and MD5 hashing as an option rather than the default. Most Linux systems now offer shadow passwords and MD5 hashing as the default. If your Linux or UNIX systems don't use shadow passwords and MD5 as the default, you need to include in your policy a requirement to enable their use.

Another aspect of password management is ensuring that users pick strong passwords. Enforcing this behavior in a UNIX environment can be difficult because many UNIX environments support multiple flavors of UNIX. Having multiple flavors lets systems managers best support their organization's mix of needs and requirements but makes defining and enforcing a set of password policies difficult because managers must customize their password procedures for every target UNIX flavor.

The best solution in a mixed environment is to use a password management tool that plugs into each target system. Although some organizations use customized scripts and Web applications as password management solutions, often as your organization grows you'll want to purchase and deploy one of the large identity and password management applications designed for the task (e.g., NetIQ's VigilEnt Password Manager—<http://www.netiq.com>, Computer Associates' eTrust Admin—<http://www.ca.com>). A

benefit of comprehensive password management solutions is that they typically offer self-service to users who need to reset their passwords. These solutions also give the Help desk the ability to delegate password management.

Vulnerability Analysis Tools

Vulnerability analysis tools give you a way to assess vulnerabilities on your network and servers. These tools, combined with a rapid response team that can reduce or eliminate vulnerabilities, enhance your vulnerability management ability.

Network vulnerability scanning requires that the scanner first detect which services are available on a server. Next, the scanner maps the ports found with the banners retrieved (a banner is the greeting a network server provides when you connect to its port) to match against a database of known applications. The scanner then uses this information to perform a series of tests against the services that are usually specific to the application the scanner determined is running. Common tests include buffer overruns and DoS attacks and can also include preprogrammed requests and responses known to result in compromises in vulnerable versions of the software.

Note: In general, randomly performing network scanning and vulnerability testing on a production server isn't advisable because the scanning might cause the server to go offline accidentally (e.g., during a test for DoS).

Examples of network vulnerability analysis tools include the open-source Nessus (the most popular network vulnerability analysis tool currently in use) and commercial scanners from companies such as Internet Security Systems (ISS—<http://www.iss.net>). Both of these tools help you detect services and find vulnerabilities across all of your UNIX servers. Depending on your needs, Nessus and other open-source tools usually provide adequate functionality. The primary differentiator between Nessus and commercial products isn't the scanning quality but rather the reporting capability. Nessus works well for single hosts or targeted network scanning, but commercial products tend to provide more readable reports for large networks.

In addition to network vulnerability analysis tools, you can use software that you install on a server and use to detect vulnerabilities in local software. This type of software is more intrusive than network vulnerability analysis tools but often provides more useful information and returns fewer false positives. Examples of open-source host vulnerability analysis tools include Tiger for UNIX and Bastille for Linux (Bastille detects and corrects many security problems). You need to understand that most host vulnerability analysis tools focus on obvious security risks such as insecure file permissions, enabled services that are known to be insecure, and unnecessary default accounts (e.g., the FTP user account). Because these areas are the most prone to result in a compromise, the vulnerability scan and subsequent hardening can dramatically increase your servers' security.

Vulnerability scanners that are specific to Web applications also exist. These tools comb a Web application, looking for potential and known security problems. Regardless of how safe a Web application programming language purports to be, you need to run these tools against Web sites before deployment and regularly thereafter to find new holes. As with host scanning software, open-source and commercial solutions exist. Most of these scanners target Common Gateway Interface (CGI) scripts, known Apache and other Web

servers' default installation files, and known problems with languages such as PHP, Active Server Pages (ASP—running under ChilliSoft ASP), and ColdFusion.

Intrusion Detection Systems

IDSs are relatively common in today's networks. IDSs fall into two categories: HIDSs and NIDSs. NIDSs receive the most media attention, although HIDSs also give systems managers a lot of power and manageability. The term HIDS actually encompasses several technologies, some of which existed before the more comprehensive concepts of NIDS and HIDS became popular. Two common HIDS components are kernel and behavior monitoring and file integrity checking.

Kernel and Behavior Monitoring

Kernel and behavior monitoring usually requires the monitored server to have a special kernel or kernel module loaded. The modified kernel monitors for suspicious activity, such as consistent attempts at unauthorized access to special privileges or files, new modules being loaded, or applications that behave differently than their normal baseline.

Although kernel and behavior monitoring can play a key role in your IDS strategy, getting this type of monitoring to work often involves a large initial time commitment. In general, you need a baseline analysis of typical behavior. Kernel and behavior monitoring IDS is more proactive than file integrity checking, which is an after-the-fact check. Kernel and behavior monitoring can help you stop an activity before it proceeds further.

File Integrity Checking

File integrity checking plays a pivotal role in a secured environment and is part of a complete IDS. You need to provide file integrity checking on all your UNIX servers. Even if you determine that a full IDS isn't required, you should still require file integrity checking on every server. When used with vigilant log monitoring, file integrity checking gives systems and security administrators timely information about attempted, in-progress, and successful system compromises.

A file integrity checker compares a file on a file system to a database containing information about the file's Last Known Good state. The checker determines whether a file has changed. If the file on disk and the file information in the database are different, an alert is generated. File integrity tools typically use a combination of hashing functions, such as MD5, to compare file contents, and metadata, such as ownership and inode number, when comparing files. One of the most common file integrity tools is Tripwire, which began life in the UNIX world but has also expanded into Windows. Several solid open-source file integrity tools exist (e.g., Advanced Intrusion Detection Environment—AIDE).

A problem with file integrity checking is that an attacker might alter the file database before the tool runs. In this case the tool won't detect or report any changes because the database was updated. To guard against this possibility, you need to store the file database on a secured remote server or on nonwriteable media such as a burned CD-ROM.

Network Security

Firewalls

Firewalls provide a security barrier between an internal network or hosts and external network devices. Two types of firewalls exist: packet filtering and application level. The traditional firewall device is packet filtering, which scans incoming IP packets and determines whether to allow a packet in or out. Packet filtering firewalls have the benefit of speed but don't understand the application protocol and therefore can't filter packets based on information such as incorrect application protocol usage (e.g., a bad HTTP header). Application level firewalls work at a higher stack layer than packet filtering firewalls and therefore understand the application protocols that are in use. However, packet filtering firewalls are slower and are more processor intensive than application level firewalls.

You use a firewall when you need a barrier between different areas of trust in your UNIX environment. An obvious example is the interface between your corporate network and the Internet. Additional areas include between departments and IT test labs.

Many UNIX OSs include firewall software in the kernel. For example, you can use Solaris and Linux as firewalls. According to the security-in-depth principle, you also need to use the packet filtering firewalls built into these systems as a way to protect servers. For example, a Solaris Oracle server should use a firewall that filters all traffic except incoming requests and outgoing responses. This practice gives the administrator more time to correct configuration errors (e.g., enabling RSH), because the service is inaccessible even if enabled.

NIDS

A NIDS constantly monitors network traffic, looking for signs that a network-based attack is in progress. NIDSs can typically monitor for known attacks (signature based) and monitor for behavior that indicates an attack might be in progress.

Even the best-configured NIDS sometimes generates false positives (i.e., alerts caused by activities mistakenly marked as attacks). This problem leads some systems managers to question whether they should use NIDS solutions for all their installations. One of NIDS's (or any IDS's) major problems is the amount of tuning necessary during the initial installation and the time necessary to monitor output. However, the general consensus is that if your organization has the resources to deploy and monitor a NIDS, you should do so. Otherwise, expend your efforts in other areas, such as increased server hardening and installing file integrity checking tools on your systems.

Insecure Communication Channels

One of the most important principles of network access is that communication channels must be secure. When considering administrative and end user remote access to a server, you need to use a cryptographically secure channel. Also, verify the identity of all parties involved in a channel. Some attacks are easy if just one network point is compromised.

An example of an attack possible because of an unsecured channel involves a compromised DNS server. (And unfortunately, the most common DNS server, BIND, has a history of vulnerabilities.) DNS drives most UNIX environments; when an

administrator connects to a remote server, he or she typically uses a domain name, such as `ldap.example.com`. If an attacker has control of the DNS server or poisoned the server's DNS cache, the attacker can redirect administrators to a server that the attacker owns. Even if the administrator were using an encrypted channel, he or she might supply a logon and password to the remote server to gain access—the attacker would then immediately have access to this information. Depending on the attack's timing, the attacker could attack the real server while the administrator was trying to determine why his or her logon failed.

In such a case, technology such as SSH or certifications can verify the remote computer. Because the attacker owns the remote computer, the SSH host key validation stage would fail, thus alerting the administrator to the problem. Because of DNS's and unverifiable network services' inherent insecurity, you need to rely on alternative verification methods to reduce your risk of attack.

Incident Response

Security incidents often occur in large environments simply because such networks offer attackers a big target. Attacks usually occur in small environments because of random selection or because an attacker was searching for a specific target (e.g., documents at a small research medical clinic). Because most sites eventually suffer a compromise, you need to have an incident response plan in place.

The first step is to define and test policies and procedures to address your response team's needs, which involves writing a security and backup policy and procedure document and assigning incident response team roles. A backup policy and procedure document is vital for incident response and disaster recovery. A major breakdown during later phases of incident response and disaster recovery is failing to quickly restore crucial systems and data to servers.

Assigning team roles is also important. If you don't define team roles now, when an incident occurs your impromptu team will probably fail to accomplish at least one core proper incident response requirement. If you don't expose a team to the predefined procedures, the team will lack coordination when an attack occurs.

In principle, incident response in UNIX environments is similar to other environments. Generally accepted responses after an attack include the following.

1. Incident identification
2. Investigation and analysis
3. Containment and remediation
4. Restoration
5. Documentation and review

Incident Identification

After an incident occurs, you need to react quickly. In some cases, such as when customer information is compromised, you must immediately notify the appropriate authorities. Then, determine whether the incident is in progress or has already occurred. Typically, an IDS notifies the information security group of an incident, a file integrity tool notes a discrepancy, or a manual log file review (e.g., of syslog's `/var/log/messages`)

shows an odd log message that the systems or security administrator reviews and determines to be from an attack. Next, you need to notify the incident response team so that the team can follow the proper incident response procedures.

The team typically makes an initial review of the system logs and state to verify that an incident occurred. You need to determine whether an incident involved an attack or resulted from a software or hardware glitch.

First, you might want to run commands such as `last`, `ps`, and `lsof` and save the output to another server. Second, you need to preserve evidence, which is problematic in a production environment because the compromised systems are often in use and taking the systems offline can be expensive. Several options exist for handling this problem. If your UNIX system is running with mirrored storage (e.g., RAID 1), you can break the array and safely store a mirror for later review. Another option is to use a UNIX tool such as `dd` to make a bit-by-bit copy of the media, in which case you need to unmount the file system or at least use the `mount` command to remount the file system as read-only. Finally, you can remove the storage media, replace it with new media, and rebuild the system.

Note: Chain of custody is important if you want to criminally prosecute an attacker. Document each stage of your incident response, including who handled any drives or storage media containing evidence.

Investigation and Analysis

At this point the incident response team is ready to investigate the incident in more detail. Depending on your business needs, the investigation might be detail oriented or mainly concerned with quickly identifying the exploited vulnerability to fix other systems. Investigations should be thorough enough to locate all affected systems, determine which systems have the vulnerability, and put into place a restoration and patch plan.

The nature of UNIX logging gives UNIX managers a good chance of finding a series of logged events across the network. If you've configured central logging, review the centralized logs for traces of odd activities (e.g., suspicious logons).

Investigation and analysis typically includes forensic analysis, which is a detailed examination of the compromised system. Forensic analysis is as much art as science and requires a high degree of skill. If your organization is large enough, train an onsite staff member in computer forensic analysis. This person needs to know how to use tools such as Brian Carrier's open-source The Sleuth Kit and Dan Farmer and Wietse Venema's The Coroner's Toolkit (TCT), as well as commercial packages.

Containment and Remediation

The next stage begins with containing the problem. You might think that this stage should come earlier; however, properly containing an incident before you determine the root causes and affected systems (which you discover during the investigation and analysis stage) is difficult if not impossible.

The team might take several steps in containment, from locking out an affected user account and ensuring that nobody is actively logged on as that user (in UNIX, locking an account doesn't affect currently logged on users or their permissions to continue working and changing files) to taking affected systems offline to ensure they don't assist an

attacker in further compromises. Taking affected systems offline is drastic but might be necessary if you can't quickly stop an attacker or remove the exploited vulnerability.

After containment is remediation, in which you plug the holes that allowed the attack. For example, if the exploited vulnerability were in the local FTP server, you'd need to patch the software if a fix were available, or disable FTP access until you developed, tested, and deployed a fix.

Restoration

In this step, restore the system to a Last Known Good state. Simply fixing the problem on the affected systems isn't advisable because the attacker might have planted a Trojan horse on the affected systems. A Trojan horse would give the attacker a back door to later reenter the system. The restoration stage requires a properly defined and executed backup strategy. Disaster recovery planning is beneficial during incident response because the affected systems are a total loss.

You need to restore the system to the point in which it wasn't vulnerable to the attack that compromised it. Otherwise, you might have a difficult time determining whether the attacker exploited the system earlier and possibly left trapdoors to get back into the server. If you can't restore the system to a nonvulnerable state or immediately correct the vulnerability, you need to perform a fresh installation, restore only the data, and lock down the vulnerable service until you properly secure the system.

Documentation and Review

Finally, you need to document the incident. The documentation needs to include the vulnerability used and an explanation for why the existing security policies and procedures didn't correct the vulnerability before the incident occurred. This information is useful for later improvements.

Disaster Recovery

The idea behind disaster recovery is to plan for worst-case scenarios. Disaster recovery is an important element of both a security policy and an incident response procedure. The security policy defines the methods and procedures that protect against and eliminate potential causes of a disaster. Disaster recovery is how you respond to those disasters.

Several tasks make disaster recovery easier: Perform regular backups of data and system configuration information, practice your disaster recovery procedure at least semiannually, and use redundant systems and remote sites that can help you automatically recover from a disaster. In addition, the most important element in disaster recovery is documentation. Documentation that details system installation, from both vendor media and backup software, must be available to the recovery team at all times. UNIX lets you easily provide documentation on most configuration settings as well, because configurations tend to be in text files that you can print and include in server documentation.

You need to codify these and other tasks into a plan. In the following sections I develop a disaster recovery procedure development cycle outline and explain how that procedure addresses a business' needs.

Areas to Protect

The first step is to define the areas that your disaster recovery plan needs to protect. This phase is often called the Vulnerability Assessment or Definition of Requirements. In my example, I focus on the systems under the Operating Systems group management. Chapter 1 defined five levels of importance for these systems, which you can use to create the following sample prioritization.

- Infrastructure servers
- Data servers
- Application servers
- Interactive servers
- Workstations

Because every site is different, no hard and fast rule exists for designating the areas that your disaster recovery plan protects. Depending on your company's workflow, you might need to reprioritize your equipment's importance specifically for disaster recovery. For example, you might want to ensure that a subset of your workstations have the same priority as your infrastructure, data, and application servers, because your users can't work without workstations.

Determine What Happened

After your core services are functional again, you need to determine what caused the disaster. Depending on your staff levels, you might be able to dedicate a second team to determine the disaster's cause, while the disaster recovery response team performs recovery. Although dedicating as many people as possible to recovery might seem like the best approach, in a crisis situation a better option can be to have a small, well-trained team dedicated to disaster recovery rather than a large team trying to attack each stage of the process.

Note: Problem determination is similar to the investigation stage during incident response.

Phases

A simple disaster recovery plan involves four stages. These stages are similar to the phases of incident response. The most important consideration in each stage is to restore crucial services as quickly as possible and restore less-used services as necessary.

Identify Disaster

First, determine what happened. Did you lose servers because of a fire? Did you lose a building? Did you lose key personnel? What services or systems are no longer functioning, and what services must you immediately restore? Identifying the disaster lets you quickly pinpoint the affected area and ensures that you don't leave your servers and equipment in danger.

Assemble Team

When an incident occurs, you need to assemble the team you've previously put together. At least one team member will probably be unavailable. This probability is why cross training and having backup team members is important.

Note: You need to realize that none of your team members might be able to reach the primary site (e.g., if an entire computing facility is lost). Thus, important functions must have an offsite backup available. Banks and hospitals regularly use offsite backup locations. These locations are usually synchronized with the data and services at the primary site. If the primary site fails, the backup site brings key resources such as billing online. Supporting an offsite location can be expensive. UNIX tools such as rsync and replication features built into most databases can ease the burden.

Recover Functions

The next step is to recover minimal service. At the minimal level of service, the company can accomplish its core functions. Most users will be frustrated with a minimal level of service, but the company can survive for a short time at this level.

As an example, let's consider a financial services company. The minimal level of service includes access to market information and trading and to a telephone system. The disaster recovery team must quickly provide access to the Internet, trading system, and customer records (in an emergency situation, a hard copy of client information might be sufficient) and ensure that a telephone system is up and able to process incoming and outgoing calls. Services that users usually consider important, such as email, wouldn't be available at this level.

Restore Full Service

The final goal is to attain a normal level of service. At this level of service, all company services are available and users can access the data and information they need.

One of the best ways to ensure a quick restoration of full service is to use the disaster recovery features built into modern backup systems. You can integrate software from companies such as Veritas into your backup strategy to allow for a quick restoration of servers. The key difference between disaster recovery software and normal backup software is the time necessary to bring a server back online. With backup software, you typically need to restore an OS from vendor media, whereas with disaster recovery software the backup lets you restore the system from scratch. Software to assist in disaster recovery is a wise investment.

Note: UNIX's dd can make a byte-by-byte copy of a disk. Many UNIX sites rely on dd the same way that Windows sites rely on tools such as Symantec's Norton Ghost. Although dd is a powerful tool, it doesn't always work if the disk being copied to isn't the same size as the disk you're copying. To provide quick and reliable restoration services, use disaster recovery software rather than dd.

Policy Compliance Monitoring

Policy compliance monitoring is complicated because it encompasses more than just ensuring that systems are installed and managed according to the security policy. Rather, policy compliance is comprehensive monitoring of everything from proper systems management to end user awareness training and programs. The three policy compliance monitoring areas I discuss are computer systems, computer usage, and user awareness and training.

Computer Systems

Monitoring policy compliance on UNIX systems involves matching various monitoring tools against the appropriate areas of your security policy. In the case of the security policy I described earlier, the guidelines cover system installation, systems management, password management, and network security. You can often categorize policy monitoring by the areas your security policy outlines. Categorizing security into sections lets you more easily automate some or all of the policy compliance monitoring.

Monitoring policy compliance during system installation typically means requiring that new servers run against an automated testing application. Most often this application is a mix of a host and network vulnerability analysis tools, which I discussed earlier, along with a series of scripts that monitor for required configuration settings that aren't within the vulnerability analysis tools' scope. For example, if your installation policy requires that all UNIX servers have a deny-by-default configuration of `tcp_wrappers`, your tool needs to ensure that `/etc/hosts.allow` and `/etc/hosts.deny` exist, their permissions are safe (e.g., file permissions mode 0400, which is very restrictive), and their contents are appropriate for the server. Monitoring policy compliance for a production server is similar. Again, the most useful tactic is to run a set of automated tests against the server on a regular basis, looking for exceptions that violate policy.

Computer Usage

Computer usage deals with how users use the computer systems you manage. An Acceptable Usage Policy (AUP) should define usage. An AUP describes appropriate and inappropriate computer system and network usage by users. Enforcing computer usage policies typically involves monitoring syslog log messages for user logons and logoffs, as well as monitoring the applications that end users use. One of the best methods of monitoring applications is to regularly run a program that logs which programs are running and notes the user account running the application. Over time you can build a database of which users typically use which applications. (This information can also assist in tasks such as performance tuning.)

User Awareness and Training

User awareness and training is the best preventative tool in your management arsenal as you try to ensure efficient and productive system use. As part of policy compliance monitoring, you need to enforce and regularly review proper training.

Conclusion

This chapter focused on the major areas of UNIX security. I developed a security policy that addressed the needs of the Operating Systems group, and I explained how to implement those needs in a UNIX environment. Throughout the chapter, I discussed several elements of incident response and disaster recovery, and I dedicated an entire section to each of these topics. Finally, I described the importance of policy compliance monitoring as it relates to users.

3

Backup and Restoration

Surprisingly, some people still don't understand the importance of a proven and reliable backup plan. Even more surprisingly, many of these people aren't those you'd expect—for example, a company's accountants, CFO, or CEO. Although these people try to cut costs, they generally understand the necessity of protecting data. Instead, systems administrators are frequently the culprits. Too often, systems administrators are so overtaxed that they lose sight of big-picture issues and instead focus only on day-to-day fires and chores.

As a systems manager you must identify your organization's backup requirements. You need to identify what data you must protect, the cost of losing the data, and how long the company can survive without access to the data. This knowledge will help you define a backup policy and determine how to allocate your backup budget.

Unfortunately, no quick and easy fix exists for backing up data. For some companies, such as ISPs, the most important elements to back up are network equipment configuration, customer databases, and billing. For others, such as hospitals, attention must focus on patient records, billing, and scheduling databases.

In this chapter I discuss managing backups and restores in a UNIX environment. The goal of this chapter isn't to introduce the technical trivia involved in using UNIX backup tools; instead, I discuss the major problems you'll face when trying to ensure that your UNIX-based data is secure and available. This chapter explains current best practices in managing backups and reviews backup technology and techniques.

Backup and Restore Policy

The first step in defining your backup strategy is to identify what data you need to protect and determine how to protect that data. To illustrate these tasks, I use an example backup and restore policy. The policy that Figure 1 shows serves as a framework for the solutions I discuss later in the chapter.

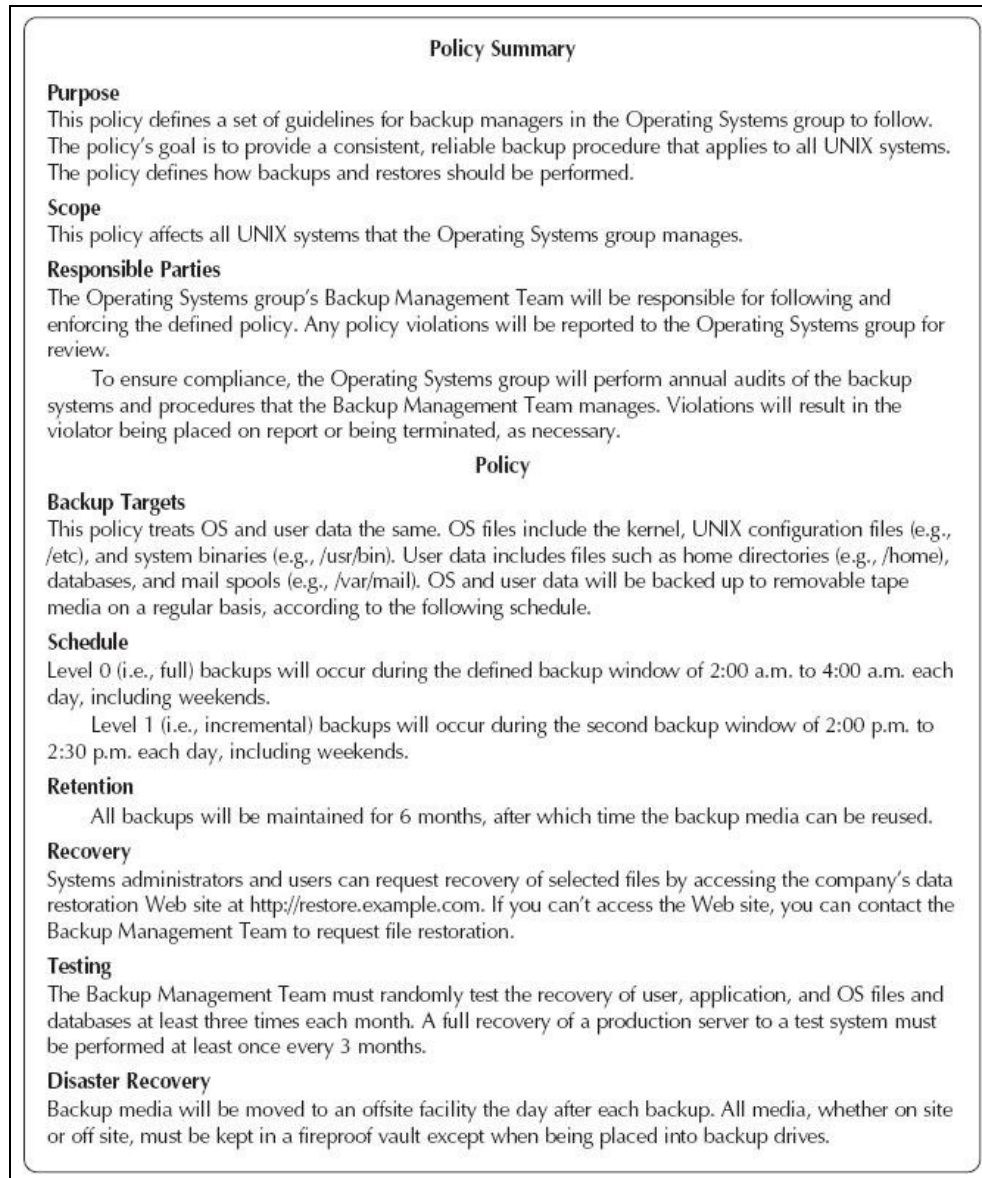


Figure 3-17. Example Backup and Restore Policy

Identifying Needs

Before you can determine optimal backup and restoration solutions, you need to understand your backup infrastructure's requirements. What kind of data do you need to back up? How will you access the data? In this section I discuss the most important questions you need to ask as you develop a strategy.

Define Restorations, Not Backups

When developing backup policies and procedures, many systems administrators and some systems managers think in terms of *backups*. However, the most important factor in managing backups is *restoration*. No one cares if a backup works. A company's IT manager or CEO will probably never ask how a particular backup went, but in case of a failure or an accidentally deleted file they will certainly want to know how soon you'll finish restoring the vice president's email.

You might think that stressing restores over backups is splitting hairs. However, if you focus only on what data you need to back up, you'll likely neglect important backup information, such as how often to back up the data or how quickly you'll be able to restore the data. Although I often use the term *backup procedure* or *backup plan*, you need to keep in mind that these terms refer to the procedure that ultimately ensures your ability to *restore* your data.

Note: I once had a client with a Web cluster that served a large number of users and therefore contained a considerable amount of data that needed protection. Unfortunately, the client had a tight budget. The client was using a very data-focused backup. He could almost immediately restore key files, in addition to restoring databases or Web files. However, the client refused to budget backups that would easily enable a bare-metal restore (i.e., a restore to bring a system back up after a complete loss of the OS and file systems). Thus, if a server failed, a bare-metal restore would take much longer than necessary.

Define Restoration Needs

Now that you know to place your focus on restores rather than backups, let's discuss a UNIX environment's restoration needs. The four major areas are:

- Application data
- Applications and OS files
- Disk and boot volumes
- Backup server software data

This list isn't one of priorities. Needs vary so widely across organizations that generically prioritizing restoration needs is difficult if not impossible. However, the list order does represent how often you might need to restore each area. You typically need to restore application files the most often and backup server software data the least often.

Application Data

The most commonly requested data type for restoration is application data. Application data includes a wide range of files, such as files from a user's home directory in `/home`, HTML and ColdFusion files for Web servers, and database tables and index files. (Some databases, such as Oracle, don't use standard UNIX file systems to store files—I discuss databases in more detail later in the chapter.)

Application data is generally easy to back up and restore. Most of the data isn't continually in use, so you can often define a backup window in which the files are available to the backup software.

Note: The backup window is the time during the day when backup applications run. In most enterprise environments you can't back up data during the workday because files are in use or the backup process places too heavy a burden on the network and systems you're backing up. Backup windows frequently occur late at night (e.g., 2:00 a.m.) because systems are least used at this time of day.

However, backing up application data isn't a trivial exercise. As you design your networks, file servers, and other UNIX infrastructure elements, you need to make backing up data as easy as possible. For example, backing up user documents is easier if all your workstations use NFS to mount user home directories rather than storing files locally. Indeed, a key reason to use NFS is to centralize files so that you can easily manage them for backup and to give users easy access. (Mounting user home directories from a central location prevents users from having to search several servers to find their files.)

Note: Fortunately for UNIX backup managers, storing application data outside of home directories or the directories used to install applications that maintain their own data files (e.g., Web servers) is rare in UNIX. In other environments, such as Windows, some applications commonly store files outside of user home directories. This scenario makes ensuring that important files are backed up difficult.

Applications and Operating System Files

The next most common files that need restoration are applications and OS files. This group includes UNIX configuration files such as `/etc/passwd`, binaries such as `/usr/bin/vi`, and application programs such as `/usr/local/apache/bin/httpd`. Unless you're upgrading or patching a system, these files change far less often than application data.

Backing up files that don't change often is easier than backing up files that change continually. For example, you can expand your backup window for applications and OS files, because the files probably won't change while they're backing up. (Of course, this procedure doesn't take into account the effect of backups on system performance.) In addition, you can back up these files less often or use differential backups more often than full backups to increase backup speed.

Note: A useful feature of most UNIX systems is the ability to restore the kernel on a running system. Most UNIX OSs don't lock the kernel file while the system is running. Instead, the kernel loads when the OS starts; the kernel file isn't needed again until the next reboot because the kernel is already loaded into memory.

The `/proc` File System

The `/proc` file system is popular on OSs such as Linux, FreeBSD, and Solaris. This file system provides a virtual window into the kernel, letting you view memory directly, set and view flags in the kernel, and perform other operations such as determining which processes are running and the environment that they're running in.

Although you can back up the `/proc` file system, doing so offers no benefit. Because the `/proc` file system is a virtual window into the UNIX OS's current state, `/proc` values will

vary across system reboots. Backing up `/proc` is akin to backing up a server's memory while the server is running.

However, most `/proc` implementations have an interesting feature. Usually, accessing files under `/proc` lets you view a server's hardware configuration. For example, viewing `/proc/cpuinfo`, which Figure 2 shows, lets you view a running Linux system's CPU information.

```
# cat /proc/cpuinfo
processor : 0
vendor_id : AuthenticAMD
cpu family : 6
model : 8
model name : AMD Athlon(tm) XP 2400+
stepping : 1
cpu MHz : 2008.531
cache size : 256 KB
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 1
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
       pse36 mmx fxsr sse syscall mmxext 3dnowext 3dnow
bogomips : 4010.80
```

Figure 3-18. Linux `/proc/cpuinfo`

Although backing up `/proc` for restoration purposes doesn't usually make sense, you can store information from several `/proc` files to easily document your server hardware and some kernel configuration parameters. One method is to build a set of scripts that can print information from key files and store this information in a central location. You can then review the information regularly to determine whether unauthorized changes have occurred, or if you simply want to know which systems are running what hardware.

Disk and Boot Volumes

A disk volume is a section on a physical disk. In the Intel world a disk volume is known as a *partition*. FreeBSD refers to disk volumes as *slices*. Although the terminology varies, the idea is the same. Disk volumes are technically an OS-level construct because the OS formats and uses them. However, considering disk volumes separately is convenient because restoring entire disk volumes is an infrequent task. To back up a disk volume, you can use a tool such as UNIX's `dump` command or a large commercial package such as IBM's Tivoli Storage Manager (TSM).

A boot volume is a special disk volume that has the kernel and possibly an OS loader in a special sector of the disk (e.g., the Master Boot Record—MBR). These volumes are used to boot the OS into memory; without them, servers wouldn't boot. (Some platforms, especially for mainframes, don't use this method. Instead, those systems require an

external OS loader. External loaders are computers that are dedicated to loading the mainframe OS.)

Note: In the past, disk volumes that never exceeded one backup tape's capacity were common. For example, if an organization could back up 4GB to a tape, the organization would never have a disk volume larger than 4GB. This situation occurred because until recently most backups weren't effective or reliable at spanning backup tapes when backing up one file system. Thus, the solution was to ensure that a file system was never larger than the tapes. Current software tends to be better at spanning backup tapes. (Exceptions exist—for example, the Advanced Maryland Automatic Network Disk Archiver—Amanda, which I discuss later, isn't effective at spanning tapes for one backup item.)

Backup Server Software Data

Backup servers are the core components in a centralized network backup, and the data that these servers store falls into a special category. Backup servers typically store license keys, tape inventory information, account names and passwords, disaster recovery data, and other information necessary to back up and restore data over the network. In addition, these servers store data regarding backup device configuration; this information can be crucial when you need to restore files.

When you provide a backup strategy, you need to develop a tested procedure to fully restore the servers that control the backup media and software. As you develop this procedure, be sure to determine what components of the backup server software must be functioning for you to begin restoring your crucial files and applications. An example problem is not being able to restore data until you've located and installed all your licensing keys. (On a related note, what happens if a normal server fails and you decide to reinstall the OS, applications, and data onto newer hardware? Will your licenses let you do so?)

Note: Having license keys handy for normal applications and backup server software is important during an emergency situation. You need to keep a hard copy of your keys in a safe location, such as an offsite bank vault or a fireproof safe. Obtaining copies of lost application installation CD-ROMs is often easier than obtaining new licenses.

You need to be intimately familiar with restoring the backup server, software, licensing, and tape inventory information to a new server in case your primary backup server fails. If a disaster occurs, your backup server will probably be affected.

Note: The possibility of being frustrated by a difficult-to-install backup package during a crisis situation means that you need to consider large commercial packages carefully. If your solution uses a proprietary backup format, ensure that you can easily read and restore from that format in case of an emergency. The tape archive (tar) format, despite its weaknesses, is a good example of an industry standard that can be read across most UNIX OSs and hardware platforms.

Understanding Backup Technology

After you know what kind of data you need to back up, you must consider the available backup technologies. Backup technology has improved greatly in the past few years. Rather than relying on single-tape solutions for each server, you can now perform massive network backups to a library of tapes and drives in one data center.

In this section I discuss some common backup technologies and their applicability for UNIX backups. Some technologies are relevant only to small UNIX shops, whereas other technologies are too expensive for any but the largest organizations. Regardless, all the technologies I discuss are proven to be the best for UNIX environments.

Types of Media

Several types of backup media are available for use in UNIX environments. The media ranges from traditional magnetic tape to disk-based backups.

Magnetic Tape

Magnetic tape has a long history, and for good reason: Magnetic tapes are reliable. You can typically drop a magnetic tape several times without damaging it, magnetic tapes survive many physical hardships (e.g., small variations in moisture) better than most other media, and magnetic tapes can hold data for decades or longer. Another factor that makes tapes popular is that they have a good price-to-capacity ratio.

Many types of magnetic tapes exist, ranging from DAT/DDS to the Linear Tape-Open (LTO) tape media family. Formats change frequently.

The relationship between disk capacity and tape capacity changes every few years. Disk capacity tends to grow gradually, usually by 10 percent or 20 percent per year, whereas tape capacity tends to remain stagnant for 2 or even 3 years then jump considerably when a new format is released.

As you develop a backup strategy based on tape (as most backup strategies are), you need to realize that the amount of disk capacity you support will eventually outgrow the tape technology's capacity. Thus, you need to ensure that you can span your file system backup to multiple tapes or that your file systems are no larger than the tape capacity. (Keep in mind that restoring data from multitape backups can be difficult. In addition, you increase the risk of losing data if one of the tapes is damaged.)

CD-ROMs and DVDs

Another type of backup media is CD-ROMs and DVDs. Although not yet popular in UNIX environments, CD-ROM- and DVD-based backups can be a viable alternative to tape-based backups for small servers.

CD-ROMs' major limiting factor is their capacity—approximately 650MB. Because of their small size, CD-ROMs are typically reserved for use at workstations for users to manually back up their own files when necessary (e.g., for their own archival use or to transport).

DVDs have a greater capacity than CD-ROMs—approximately 4GB. Although DVDs are small compared with modern tape capacity, 4GB is sufficient to back up UNIX servers in small offices and remote sites. Such servers typically store a limited amount of

data; providing a DVD-based backup solution for user data is an easy way to ensure that backups are available to those sites.

Although CD-ROM and DVD backups are popular in the PC and Windows world, UNIX environments haven't embraced CD-ROMs and DVDs. You might prefer to just use tape-based backups, because all UNIX backup software supports tapes. Finding a backup application that natively supports CD-ROMs and DVDs can be difficult.

Disk-Based

The relationship between disk capacity and tape capacity is constantly in flux. Tape capacity is sometimes larger than an average server's disk capacity. However, disk capacity currently exceeds tape capacity. Because of the gap between disk and tape capacities, a trend has developed to use large disks to back up other disks. In the past, this strategy wouldn't have worked because of disk reliability issues. (The risk was that the disk to which data was being backed up would fail when you needed the data for a restore, thus negating the backup plan.) Most companies weren't willing to incur the risk of a disk-based backup system. But the strategy has gained acceptance as disk reliability has improved.

Disk-based backup systems typically include one or more servers that are dedicated to network backups. These servers use RAID 0+1 to manage disks, which ensures high reliability of the storage even if multiple drives fail. The servers then use various techniques, including NFS and remote agents, to back up servers and databases over the network to the disks.

Although disk-based backup systems are becoming more common, they're still relatively unproven compared with tape systems. No best practices exist for disk-based backups. However, anecdotal evidence indicates that properly deployed disk-based backup systems can be effective and reliable.

Note: An important component in ensuring the success of a disk-based backup solution is the ability to remove drives from the backup server for offsite storage. Storing disks off site dramatically increases the likelihood of having a good backup in case the backup server or the entire server room is damaged (e.g., by fire). Don't restrict this practice to disk-based backup systems; store all backup media off site whenever possible.

Snapshots

A snapshot is a point-in-time copy of data. You can't obtain a snapshot during a regular backup unless you ensure that all services are disabled when the backup runs, which is difficult to accomplish in a production environment. Although snapshots aren't a type of media, a snapshot can be an effective tool in helping you decide how to back up and store data.

One of the most common uses of snapshots is to back up live databases. The idea is simple: A live database's storage splits, with one storage device continuing to serve the database and the second device—which has an exact copy of the database from the moment the split occurred—going offline. You can duplicate this technique by breaking a RAID 1 array and backing up one disk while the other disk continues to serve requests.

Snapshots offer several benefits. For example, a snapshot is usually transparent to the applications using the storage. Instead of forcing a database to stop processing during backup, you simply take a snapshot and back up the copy.

Snapshot technology is becoming common in Network Attached Storage (NAS) devices. A NAS device creates a snapshot of its file system either on a scheduled basis or through manual administrator intervention. (Some backup software also offers snapshot capabilities.)

Some UNIX OSs (e.g., FreeBSD) have considered using snapshot-like capabilities in their file systems. If your UNIX OS supports snapshots, you'll probably find them to be an effective backup tool. To determine whether your OS supports file system snapshots, read your OS's documentation or contact your OS vendor.

Backup Drives

Other than disk-based backups, all backup technologies share the common attribute that you insert media into and take media out of the backup drive. Even in the case of disk-based backups, you might be able to remove the disk drives to place them off site for safe storage.

In this section I discuss the three major categories of backup hardware: single media, autoloaders, and jukeboxes. I focus on tape backup drives because of their prevalence in UNIX environments. However, all backup media use the technologies I discuss.

Single Tape

The most common type of tape drive is a drive that loads manually and accepts only one tape. An administrator must physically insert a tape into this type of drive. Many modern drives let you use software commands to eject tapes.

Single tape backup drives are inexpensive in terms of hardware but quite expensive in terms of human resources. Because someone must be available to rotate tapes after a completed backup, or even to swap tapes during a multitape backup, these drives can be a hindrance during large backups.

Consider a drive that supports DDS4. This drive can comfortably back up one general-purpose application server. After the backup a human operator must retrieve the tape, locate the tape for the next night's backup, and insert the new tape. If you wanted to back up three servers, using one tape for each backup, the backup operator would need to be available not only after the entire backup completed but also after each server backed up.

Autoloaders

An autoloader is a tape drive that accepts only one tape but that loads automatically. An autoloader typically has a loading magazine that contains several tapes. When a tape fills, the next tape in line loads into the tape drive.

Autoloaders are a useful solution to the manual-labor problem of tape drives that accept only one tape. Although autoloaders are expensive, their cost is usually justified except perhaps in a very small environment. If your backup method requires you to swap tapes during the backup, you need to consider investing in an autoloader.

Autoloaders are also helpful because they can load new media into the drive when most companies have little or no staff on site (e.g., late at night). Having an autoloader can eliminate the need for an onsite backup operator when the backup runs.

Jukeboxes

A jukebox is basically a more comprehensive autoloading backup device. Autoloaders are tape drives that have access to more than one tape but only a limited number of total tapes (eight tapes is common). A jukebox contains an autoloader (which is sometimes a robotic arm), frequently multiple tape drives that can be used concurrently, and a library of tapes.

Jukeboxes are more intelligent backup drives than autoloaders are. Typically, a jukebox can randomly select which tape to insert into one or more tape drives to complete backups or restore requests. With the appropriate software, a jukebox can provide a comprehensive backup management solution. Jukeboxes can completely automate backup management.

Software

Backup software binds backup technologies together. Backup software runs on one or possibly several servers. Depending on your needs, backup software can offer simple backup features (e.g., using the UNIX dump command) or advanced backup and disaster recovery capabilities (e.g., TSM, Veritas Software's NetBackup).

Of all the backup technologies (e.g., backup drives, media—primarily dictated by your backup drives, tape library systems), you usually must live with your backup software the longest. Establishing an effective backup infrastructure for your environment can take months or even years. Although hardware gets a lot of attention, hardware becomes outdated quickly. Fortunately, you can easily replace aging backup hardware with newer, more advanced, and higher capacity tape drives. Replacing backup software is a more difficult task. Most advanced backup software requires that you load agents on target systems and that you train your staff and perhaps even your users to use the software in a certain way. In addition, configuring and managing backup software usually involves excessive training and experience. Retraining your staff and reconfiguring servers to use another software solution is difficult and expensive. Because your company will probably use its backup software solution for many years, you need to select the software carefully.

Note: If you use backup hardware or software technology that uses a proprietary format, you might face vendor or even version lock-in. Don't avoid proprietary formats just because they're closed standards, but as you make your purchasing decisions be aware that you might need to accept a proprietary format.

Protecting Your Backup Media

As you devise your backup procedures, you need to consider how to protect your backup media. Choosing the media, drives, and software is important, but those choices are irrelevant if you can't access the data you need to restore. Two common causes of not being able to restore data are lost media (stolen or not organized) and destroyed media.

Organizing Media

Imagine that you need to find and read a particular book within a few days. You go the library and find, to your amazement, that the Dewey Decimal System is no longer used.

Books are still roughly grouped together by topic, but no standard methodology exists for organizing the books. You hope that you'll find the book you want in the appropriate section, but you have no way to verify whether the library has the book. This situation probably sounds exceedingly frustrating. Unfortunately, many companies use a similar system for backup tape storage.

Many administrators who maintain backup media simply toss backup tapes into a box labeled *Backup Tapes*. Sometimes several boxes exist. Often, the only label on a tape is the day of the backup and the tape number (e.g., Friday—3). Although this method works (i.e., you can probably find the tape you want if you dig through the box), the method is inefficient. Suppose you needed to restore data within minutes or your company would lose incoming sales. In such a case, you'd want an organized method for storing tapes. (A simple system is to use a well-defined labeling methodology, as I discuss in the following section, and to physically separate media based on server or day of week.)

If your company stores several backup tapes, you need to devise a methodology for labeling, categorizing, and storing your media. Store tapes from different backups separately. For example, don't mix OS backup media with the financial department's Oracle database backup media. Most sites have a significant number of backup media; mixing different media sets can dramatically increase the amount of manual labor necessary to find a tape in an emergency. Spending 10 minutes searching for a tape to insert into the drive for a nightly backup isn't significant, but time is crucial when a key server fails. At a minimum, you need to separate your backups into different classes (e.g., application data, OS files).

Label Media

You also need to label your media appropriately. The labels should provide enough information to let someone restoring data quickly identify the correct media. For example, Figure 3 shows the label for a backup tape used to back up /usr on the second Friday. The label shouldn't include too much information, but it should contain enough information that you could use the tape even if your electronic tape inventory was destroyed. (A good goal is to be able to restore a system using only a tape drive, a tape, and a SCSI cable.)

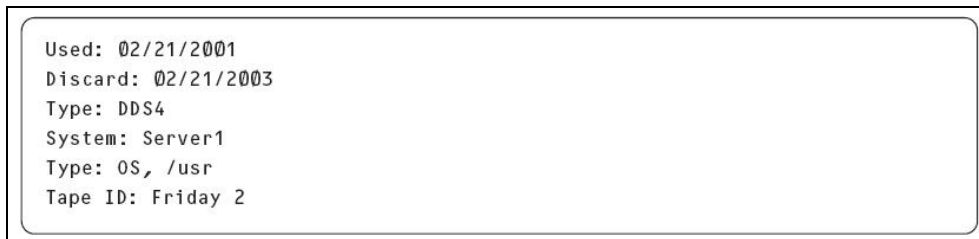


Figure 3-19. Backup Tape Label

An important element of the tape label is the First Used and Discard fields. These dates show when the tape was first used and when its expected lifespan is over. All tapes have a limited lifespan, which you shouldn't exceed regardless of how well a tape seems to be working. Many commercial backup packages track tape expiration and alert you when you need to discard a tape. However, recording the expiration date on tapes is a good idea because you don't have to use the backup program to know when to discard media.

The Type field indicates the backup device technology that can read and write to the tape. This field can be crucial in a crisis situation, such as during disaster recovery. If you need

to quickly locate a backup device to restore files from your existing set of backup tapes, you won't have much time to determine what kinds of tapes you have.

If your tapes won't hold as much information as Figure 3 shows, consider labeling your tapes with an ID number and the Type field and maintaining separate documentation that includes the full tape information. Ensure that this documentation is available to backup administrators even if your servers are unavailable. If possible, label your tapes electronically. For example, the GNU tar command lets you use the `--label` option to place a label at the beginning of the backup. When tar later reads the tape, this label displays. The electronic label lets you access the tape information even if the physical label falls off. The physical label is still important, however; electronic labeling is an additional method rather than a replacement.

Saving Media from Destruction

Best practices dictate that you secure your backup media's storage. At a minimum, you should use a fireproof safe that is difficult to move (and therefore difficult to steal). Compared with the difficulty of stealing a server or attacking a network from the Internet, stealing backup tapes is incredibly simple. In many cases, a thief can simply grab your latest backup tape to access your customer database. Someone walking into a server room and grabbing a tape might seem unlikely, but such a theft would be simple: Many server rooms are empty most of the time, and backup tapes are often in a box next to the backup drive.

Note: Securing your backups also means restricting access to the backup tape drive (e.g., placing a backup server in a locked server cage). In addition, you need to lock your server room. These actions place two extra obstacles in a thief's path.

Even if you store your backup tapes in a safe, you need to consider storing them off site as well. Consider that you'll need to use your backup tapes in any type of disaster recovery and that the worst case that disaster recovery protects against is the complete loss of the facility. If your facility is lost and you store your backup tapes on site, your backup tapes are also lost. To prevent losing media in case of a local disaster, you can move your backup media off site or you can leave the original backup media on site for easy access but maintain copies off site.

An effective offsite storage solution is to share backup media between facilities within your company. For example, you might swap tapes with another company office building, preferably on a different campus. The best offsite storage option is a company office building in a different city or state. The ideal remote site is safe (i.e., uses a fireproof safe to store tapes), is geographically remote to ensure that a local disaster won't affect both sites, and has trusted personnel. Implementing this type of storage is difficult but feasible. Suggested steps include the following:

1. Identify a remote company site that offers you a stable environment for storing backup tapes.
2. On a regular basis, ship the latest backup tape (via FedEx or courier) to the remote site.
3. A few times a year, retrieve an old tape and perform a test restore.

Note: Businesses with only one location can't exchange backup media with another site. In this situation you might contract with an outside company that is willing to securely store your backup media. Alternatively, you can use a safe location such as a bank vault to store your media.

The Art of Scheduling

An important part of devising an optimal backup strategy is to determine your scheduling needs. Some environments have long off-hours times that you can use for backing up databases, servers, and user files (e.g., a 9:00 a.m. to 5:00 p.m. medical clinic). Others types of companies, such as many financial institutions, don't have obvious backup windows.

Scheduling Backups

A major issue that affects backup scheduling is that performing a backup places a large burden on the system you're backing up. Backups are resource heavy; during a full backup, a system can be unusable if the backup fully utilizes the machine to perform the quickest backup possible. Resource use isn't only a factor for the backup software. In many situations, such as for certain databases (i.e., open-source databases such as MySQL and PostgreSQL), you must first back up the database to a series of files that the backup program can read. This action creates at least two major performance hits on the server: the database backup to files that the backup software can read and the subsequent backup of these and other files over the network.

Because backups affect the performance of the servers being backed up, you need to establish a workable backup window. A backup window is the scheduled time for backups. When establishing a backup window, keep in mind that the window needs to occur when the systems are least busy and can most quickly respond to the backup software's request for data. In most situations the backup window is late at night and spans several hours. For example, a bank might have a backup window between 1:00 a.m. and 3:00 a.m.

If a server is slow to respond, the backup software might have to slow down the tape drive speed. Tape drives can typically run at two or three speeds; the maximum speed is usually much faster than the other speeds. If the backup software slows down the tape drive speed, the backup will slow considerably. And if several servers slow down the backup, the backup might run past the backup window.

A common problem that backup managers make when scheduling backup windows is forgetting about other jobs being processed during the window. UNIX users often use the `at` and `cron` commands to schedule work on servers. Thus, multiple jobs that interfere with one another might be scheduled to run simultaneously. Consider a database server: A database administrator might schedule several reindexing jobs to run late at night when the server is least busy. Likewise, most backup managers schedule backups to run late at night. If the reindexing and the backup run concurrently, the backup might run slowly or back up incorrect data.

You need a well-documented and well-publicized backup schedule. Other managers and systems administrators must understand that running automated jobs during the backup window will slow down their jobs and the backup.

Note: You can use system performance tools, such as System V's system activity reporter (sar), to determine when your server is least used. This information is important in scheduling your backup window.

Scheduling Restores

Imagine that you're the backup manager for a large retail operation. The point of sale (POS) terminals are connected to a central processing and database server, and the POS terminals won't operate without the main server. At 8:00 a.m. one morning you discover that someone deleted a major set of records on the server. The system is down and customers are irate. You immediately begin restoring the data from your backup. At 8:30 a.m. the restore is still running; you can hear the tape streaming and occasionally stopping, rewinding, and proceeding again. The server indicates high utilization, especially on the disks. You're confused about the lengthy restore time because the backup was complete in 5 minutes.

Unfortunately, restores always take longer than backups. One reason is that disks write slower than they read. Another problem to consider is that you often must perform a second restore. For example, sometimes after you finish restoring a set of files a user discovers that he or she needs additional files restored. You might want to double the time when you estimate the time necessary to perform a restore.

Note: Because restores can take a long time, you need to provide not only the ability to restore data but also built-in redundancy on your network. In the example scenario I discuss, the best solution would be to have a redundant, always-available POS server. Putting this server into action could enable a timelier restore to the main server.

One way to inform users of restore times is to document a table of estimated restore times for various data types. To create such a table, you need to consider the amount of time necessary to perform the restore, as well as the time delay before you'll have the time and resources to start the job. Most IT departments are busy and can immediately handle only the most urgent requests.

Note: Automated tape backup and restore software is important in an enterprise environment. Because of the number of users in an enterprise, most companies will have several restore requests every day. Unless you can dedicate someone to manually restore files for each request, you need to automate the process. One solution is to give users an easily accessible front-end method for requesting restores (e.g., a Web application). Depending on your users' sophistication level, restore requests might go straight to the backup and restore software for processing, or requests might go through a technician who determines which files to restore.

Automating your systems administration tasks reduces errors and increases responsiveness to requests. I discuss automation in more detail in Chapter 8.

Decentralized vs. Centralized Backups

The two main types of backups are decentralized and centralized. Many companies with remote sites use the decentralized model. As Figure 4 shows, in the decentralized model each server has its own backup drives and media, and each server is individually scheduled to perform backups. Even if you develop one set of applications to manage all your server backups, you're still using the decentralized model.

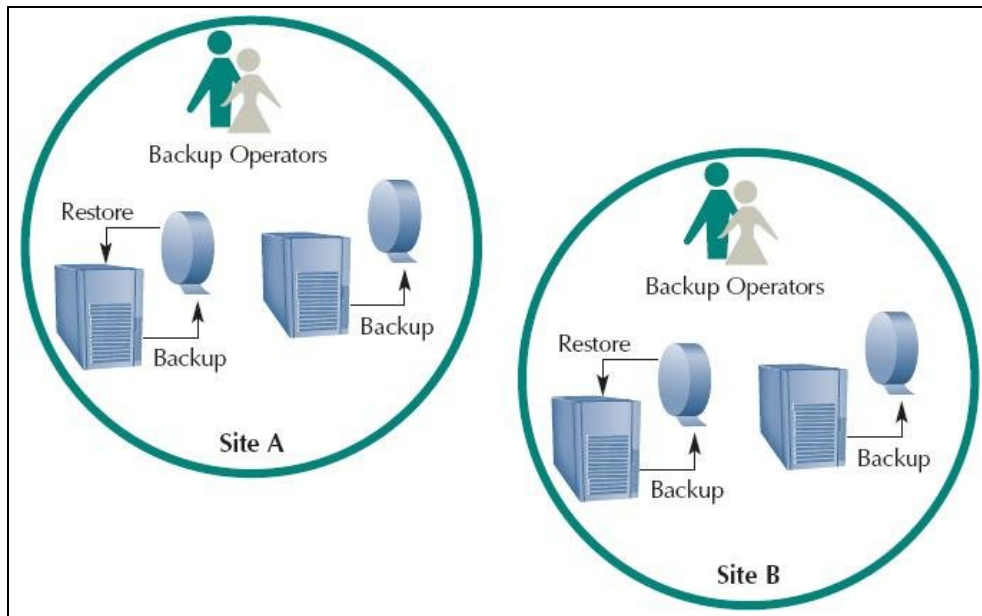


Figure 3-20. Decentralized Backup

The decentralized model has pros and cons. One pro is that a decentralized backup can dramatically reduce the use of WANs for remote site backups. Depending on the cost of maintaining your WAN, using the WAN for another purpose (e.g., for bank transactions) might be more cost effective than using it for backups. In many situations, however, this cost effectiveness is illusory. A problem with decentralized backups is that you might dramatically increase your support costs without realizing you're doing so. For example, suppose you have 16 sites, each with its own backup solution. (Most decentralized backups use different software and media at different sites simply because the solutions were implemented at different times.) You need at least one trained person at each site to operate the backup. In addition, you can't easily move people between sites because most of the sites have their own backup software and technology.

As Figure 5 shows, in the centralized model one set of software and hardware backs up data from all the company servers. This method is called a network backup: All backups take place over the network to a central backup media source.

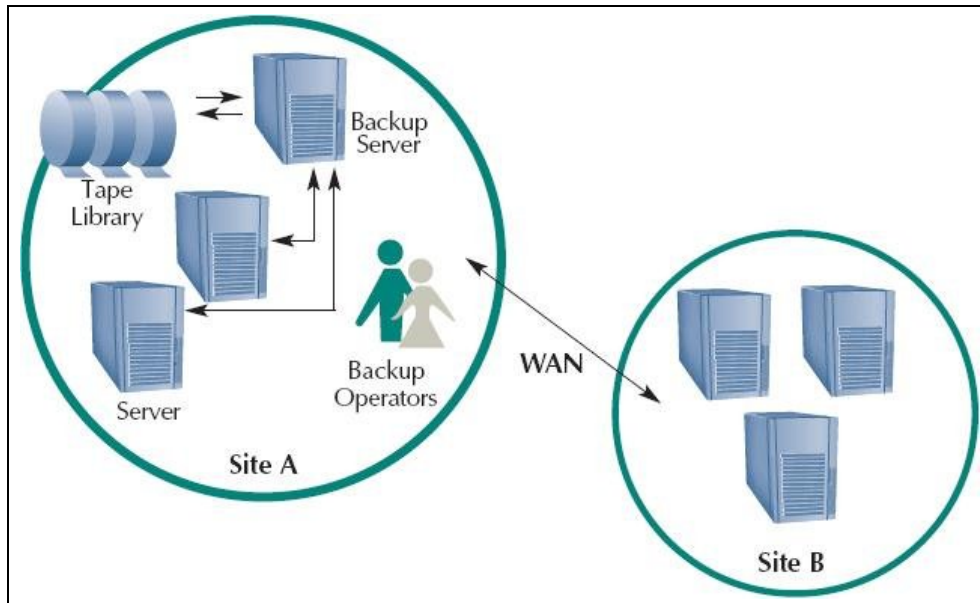


Figure 3-21. Centralized Backup

Because you have only one set of software and hardware in a centralized backup of local and remote sites, the amount of excess capacity is typically reduced and your costs are therefore lower than for a decentralized backup. (Many sites that use the decentralized strategy have more capacity than necessary; other sites constantly struggle because of capacity overuse.) The centralized backup strategy also lets you spread your backup costs across all sites, rather than each site carrying its own load. In addition, having a centralized backup site can ease the burden of maintaining copies of backup media at offsite locations.

Before you decide to centralize your backups, you need to consider some political issues. For example, many sites prefer to maintain their own backups. One reason might be that a remote site has confidential or crucial data that the administrator considers too important for others to safeguard. Another reason might be that a site had bad past experiences with centralized backups. Your goal is to convince the remote site that a centralized backup is sufficient for its needs.

Another consideration in employing a centralized backup is that a large amount of traffic will traverse the LAN (and possibly the WAN, if you back up remote sites). This traffic can consume network resources and might considerably slow network access during the backup.

A common solution to network congestion during backups is to maintain two networks: a production network and a network dedicated to backups (e.g., a Storage Area Network—SAN). The production network hosts the company's main traffic, such as email access, file transfers, and database queries. The backup network hosts only traffic for performing server and possibly workstation backups.

Backup Automation: The Key to Survival

I've already stressed that automating your backups is a good idea. Here, I define the benefits of backup automation. I also discuss techniques for backup automation, goals to work toward, and solutions you probably already use that also work in UNIX environments.

A main benefit of automating backups is avoiding the problems you might encounter if you don't use this method. One of the biggest problems in manually performing backups is that humans often make the simplest tasks more complex than necessary. Common human errors include using the wrong tape, handling media roughly and thus damaging it, and losing media. (Losing media is easier than you might think; a backup operator can easily become distracted while reorganizing a set of backup media and misplace a tape.) Automating your backups reduces the possibility of human error affecting your backups' reliability.

Another problem in manually performing backups is that backup operators are often too busy and sometimes too bored to perform the backup correctly. A backup operator who is too busy might neglect backups, sometimes intentionally. Administrators' performance evaluations often depend on how well they perform their main tasks, such as fighting fires. Day-to-day management such as performing backups often gets low priority even from those who assign tasks. A backup operator who is bored might get tired of constantly swapping tapes and performing test restores. In a large backup, the administrator might need to run several scripts and swap several tapes on a regular basis. The administrator might soon become apathetic about whether the tapes are rotated on schedule.

Computers are designed to perform routine, repetitious work. Most humans don't enjoy such tasks. Automating your backups removes one of the major causes of backup failure—humans.

Custom Automation Solutions

UNIX scripting is a good tool for automating backups. Because UNIX commands operate consistently (e.g., indicating that an error occurred based on a command's return value), building a set of UNIX scripts to handle backups to a local tape drive is simple. In this section I explain how to build such a script and where to apply it, and I examine the script's limitations. Although UNIX consistently provides a file-based interface to devices, no convention exists for naming those devices. My example uses a Linux-based backup.

Linux, like UNIX, offers many native tools for performing backups (e.g., `dump`, `tar`, `cpio`, `pax`). The `dump` command is common for performing volume-level backups. This command is versatile and quick; it efficiently returns UNIX to a production state after a major disk loss. However, `dump` doesn't work well across UNIX flavors because the command backs up a file system rather than files.

This distinction might not seem important. However, loading a backup to a system running another OS is common; if that OS can't read the file system format of the backed up UNIX version, you might be stuck unless you can restore the original OS. File-level tools, such as `tar`, `cpio`, and `pax`, don't have this limitation. I use `tar` in my example because of its popularity. Most UNIX users and administrators are familiar with `tar` because most files are distributed in the tar format (i.e., `.tar` or `.tgz` extensions).

Most backup scripts follow the following format:

1. Rewind the tape.
2. Back up the files to the tape.
3. Verify the backup.
4. Identify problems and notify the administrator.
5. Eject the tape.

The magnetic tape (`mt`) command rewinds the tape device (Step 1) on most, if not all, flavors of UNIX.

```
# mt f /dev/st0 rewind
```

When using the `mt` command, you also need to detect whether a tape is actually in the drive. You can use the string “The device is offline” to indicate in the command’s output that the drive doesn’t contain a tape.

```
# mt f /dev/st0 rewind | grep "The device is offline"
```

The `tar` command performs the backup operation (Step2).

```
# tar cpf /dev/st0 /
```

This command backs up all the files under the root directory (i.e., all the files on the server) to the tape device `/dev/st0`. You can add instructions for the command to return a non-zero error code if `tar` fails.

```
if tar cpf /dev/st0 /; then
    ERROR-CONDITION
fi
```

In this example, the `ERROR-CONDITION` statement would be custom code that responds to an error from the `tar` command. A useful response is to email an alert to a backup administrator (Step 4).

```
if tar cpf /dev/st0 /; then
    echo "Backup Failed" | mail -s "`date` Backup Failure" root
    exit 1
fi
```

You need to explicitly handle the errors that generate from the programs your backup scripts use. Otherwise, you might have nothing but blank tapes even though you think your backups are running successfully.

GNU `tar` is Linux’s version of `tar`; GNU `tar`’s `-W` option causes `tar` to verify any archive that it creates (Step 3).

```
if tar cp W f /dev/st0 /; then
    echo "Backup Failed" | mail -s "`date` Backup Failure" root
    exit 1
fi
```

Finally, the tape ejects (Step 5).

```
# mt f /dev/st0 offline
```

You can combine all these commands into one script that runs from cron during the backup window. You can also add other commands, such as dumping database volumes

to a file for backup. My example script is flexible and gives administrators hands-on backup control.

Unfortunately, using a custom-developed script creates problems. Building a script to fully react to all error conditions is difficult. In many situations tar returns a non-zero error code after exiting for a normal failure (e.g., occasional files changing while tar is reading them). Although you don't want most files to change, you can expect that some (e.g., log files) will do so.

Most internally developed scripts err on the side of caution: They consider any error return from tar as a fatal error and therefore alert the backup administrator. This situation might seem appropriate but actually causes problems because backup administrators eventually ignore repetitive errors. If an important backup error condition occurs, an administrator might overlook it. Thus you've lost one of the main benefits of backup automation: letting the server handle repetitive tasks and using human intelligence to troubleshoot extraordinary problems.

Another problem is that my example script supports only one tape. (Although some UNIX systems, such as Linux, provide tools to control autoloaders from scripts, most UNIX systems don't.) Many servers have huge disks that can be significantly larger than the backup drive capacity. Thus, many servers require multiple tapes for a full backup. The only solution for a script such as the one in my example would be to break the backup into sizes roughly equal to the tape capacity. For example, you might run the script with tar backing up /, /usr, and /var, then on another tape backing up /home. A backup administrator would probably need to be on site to swap tapes. Therefore, custom scripts typically don't scale well.

Using internally developed scripts to perform network backups can be equally difficult. A common solution is to use Secure Shell (SSH) to log on to remote servers and back up an archive to stdout.

```
# ssh root@remoteserver /usr/bin/tar czf - / > /dev/st0
```

This command logs on to the remote server as user root and backs up all files, starting at /. The command sends tar's output to stdout instead of to a file. The redirection command (i.e., >) then sends this output over the SSH connection, to the local tape device /dev/st0. You can use this command to back up several servers sequentially.

```
# ssh root@remoteserver /usr/bin/tar czf - / > /dev/nst0
# ssh root@remoteserver2 /usr/bin/tar czf - / > /dev/nst0
# ssh root@remoteserver3 /usr/bin/tar czf - / > /dev/nst0
```

The /dev/nst0 command lets you put multiple tar archives on one tape. This approach works well for small backups, but sequentially backing up several servers in a large production environment is short sighted. The operation can take several hours if you back up servers that contain a large number of local files, such as file servers and mail servers with local spool directories.

More advanced backup solutions instead spool the data to be backed up to a local disk; the local data is then written to tape. This method can considerably shorten the backup window, even if the backup server requires several hours to send all the backed up data to tape.

Open-Source and Commercial Solutions

Because custom scripts don't always satisfy a production environment, you need to consider other solutions. Several open-source and commercial solutions exist.

An open-source application that provides a comprehensive backup solution is Amanda. Amanda supports autoloading and has an error-detection capability and local backup spools so that server backups can run in parallel. Unfortunately, Amanda has some limitations. The application doesn't span tapes—which doesn't mean that Amanda can't perform a multitape backup, only that it can't span one backup target (e.g., a dump of /usr on a remote server) across multiple tapes. Amanda can be difficult to use, especially because no advanced GUI exists for it. Many USENIX administrators and managers downplay Amanda's complexity, but having a usable graphical interface that lets you quickly perform nonroutine tasks is a good idea. (Having a GUI doesn't affect routine tasks because most routine tasks are automated, but junior backup administrators and end users can benefit from GUIs.)

Commercial software, such as NetBackup, TSM, and Legato Systems' NetWorker, also offers an interesting mix of features. NetBackup is an agent-based backup solution in which remote clients send their data to be backed up rather than NetBackup pulling the information. NetBackup supports autoloading and tape libraries and lets you run backups at any time for any client. A useful feature of software such as NetBackup is the ability to plug in application-specific backup agents for databases such as Oracle.

Monitor Problems Instead of Successes

Earlier I mentioned that a major drawback of using solutions such as custom-scripted backup software is that you can easily send too many alerts to backup administrators. Over time, administrators start to ignore excessive alerts. Administrators might erroneously think that backup errors are normal.

When you develop a backup solution, whether internally developed, based on an open-source package such as Amanda, or relying on a commercial application such as TSM, you need to carefully tweak and filter the backup system's output so that the administrator receives only true errors. Otherwise the administrator might start ignoring errors and will miss any real errors that occur, thus making your backups unreliable.

Understanding Database Backups

Databases are common on networks, and every backup administrator must deal with them. But unlike most other application types, databases are difficult to back up.

Novice backup administrators often naïvely try to back up a database in one fell swoop. Depending on the software used, the backup administrator might have a reliable backup but only be able to easily restore the entire database management system (DBMS) instead of individual databases or tables. In such a case, the backup administrator fell into the trap of focusing on backups rather than restores. A full restore of an entire database is rarely necessary. In most cases, you need to restore only specific tables or individual databases. If the database administrator had focused on restores rather than backups, he or she would have realized the need to use a solution that let him or her restore everything from the entire DBMS to specific tables or perhaps even rows.

One reason for the difficulty in backing up databases is that databases often have their own disk volumes allocated rather than using a file system native to the UNIX OS being used. Databases tend to manage their own disk volumes simply because of speed. Databases rarely need access to the high-level services that a UNIX file system provides. Instead, databases prefer to handle their data at the block layer, even managing their own disk caches. This situation, combined with the fact that you can't reliably back up open database files, means that backing up an active database is difficult.

Performing the Backup

Three main solutions exist for backing up databases. You can use database vendor-supplied backup software or agents, stop the database software while the backup runs, or take snapshots of the disk volume.

Agent-Based Backups

Using database vendor-supplied backup software generally lets you back up a database while it's running. Agents that back up databases often disable updates to tables; instead, all writes go to a series of log files while the database tables and index files back up. When the backup is complete, the agent reenables writes to the tables and flushes the logs. Agents are an effective strategy but can be expensive.

Some open-source databases, such as MySQL, have pseudo-agents that let you back up a server while it's running. This capability is important because you might be able to deploy similar solutions to other DBMSs. MySQL has a backup tool named `mysqldump`. Using `mysqldump` without precautions doesn't work because the tables being backed up might change while the backup is running. However, `mysqldump` lets you use the `—lock=tables` directive to write-lock tables while they're backing up. (While the tables are locked, other applications must wait to update any tables being backed up.)

```
# mysqldump lock tables dbname > dbname.sql
```

Stopping the Database

Stopping the database is another option. If the database isn't running, it can't update the tables—and you can therefore directly back up the tables. This strategy assumes that your backup software can read the disk volume. If your software can't read the disk volume as a file system, you'll need to perform a disk volume-level backup, which involves using a tool such as `dump` to perform a bit-by-bit backup. Advantages of this method are that you don't need an agent and you don't need to understand what's on the disk volume to back it up. However, this kind of backup often limits you to performing only full database restores. If you want to restore only a set of rows or even a table, you still must restore the full database.

Taking Snapshots or Mirrored Backups

Finally, you can take a snapshot. This technique, which I discussed earlier, is effective for databases. The idea is to place the database disk volume or files on a mirrored array. When you need to back up the system, simply take one of the members out of the array and back up that member. After the backup is complete, reattach the member to the array. The array then rebuilds the member so that the member matches the other array members' updated states. Although this strategy is effective, the array rebuild can consume considerable resources for several minutes to an hour after you rebuild the

array. Your backup server might experience decreased performance as a result, which isn't acceptable in some environments.

A similar solution to taking snapshots is to use a replication peer as a backup. In this technique your primary server constantly replicates changes to a slave backup server. When you need a backup, the database on the replication server stops, replication is disabled, and the replication slave is backed up. (Many DBMS packages, such as MySQL and Oracle, include replication. Read your documentation or contact your DBMS vendor to learn how to enable this feature.)

Testing Your Procedures

Validating your backups is an important step in the backup and restore process. A lack of error reports from the backup application doesn't necessarily mean that the job completed successfully. As a backup manager your job is not only to ensure proper backups but also to ensure that you can perform proper restores. You must evaluate the technology used, as well as the procedures that dictate how restores are performed.

Perhaps the most crucial step in backup management is to test regularly. Although backup technology is reliable, problems sometimes occur. As I discussed earlier, a major cause of problems is human error. In many cases the human error is indirect and difficult to detect until too late.

For example, suppose your backups have been running without error for 6 months. All database systems, mail servers, and infrastructure devices are backing up each night—and in some cases every 3 to 4 hours. One day a new executive loses key files for an important sales meeting. When you search for the files to restore, you can't find the executive's home directory. You realize that when you configured the backup, you set the configuration to include only explicitly configured directories (e.g., /home). The Operating Systems group recently added a new directory, /home2, for new users. The backup system configuration was based on an incorrect assumption (i.e., that all user files were under /home). Your human error now prevents you from restoring key files.

Human errors will inevitably occur. To find these errors, you must use real-world test cases to test your backup procedures. In addition, your change management policy needs to include the requirement that changes to servers are properly documented and submitted to the group in charge of performing backups and test restores.

How to Test

One way to test your backup procedures is to regularly perform what I call a blind test. In a blind test, you or a random user selects a random file for a restore. This approach isn't technologically savvy, but it gives you a good cross-section from your environment on a regular basis to test your backup procedures.

Another useful technique is to not allow the backup administrator or manager to help or offer advice during the backup test. This technique tests not only the technology and techniques you use but also your staff's training. You need at least three staff members trained to perform full restores of all systems. These staff members must know how to reinstall OSs, applications, and data in case of a disaster. If you have offsite backup storage, you need to also train the offsite personnel. In the event of a large-scale disaster, you might need offsite personnel to assist in a restore at your site or at the remote site.

Operating Systems

Although the blind test is effective for assessing your ability to restore application data or entire applications, this test doesn't work for testing OS file restoration—particularly on production servers. You need an alternative method to test your ability to quickly restore an OS's files or entire disk volumes.

To test your ability to restore OSs, you must consider several test cases. For example: Can you easily restore a deleted UNIX kernel? What if someone shuts down the server before you restore the kernel? Can you perform a bare-metal restore (i.e., can you use the backup to restore a server completely from scratch)? Do you need to simply boot a CD-ROM that brings the backup software up and begins the restore? Or do you need to first install the OS to attain some minimal functional level, install the backup software, then restore the OS? You need to incorporate these and other test cases into your larger testing policy.

Conclusion

In this chapter I discussed performing backups and restores in complex UNIX environments. You need to properly identify your restoration requirements before you can implement a useful and comprehensive backup policy. Scheduling ensures that you don't adversely affect your UNIX network's performance. Most environments benefit from a centralized backup solution; in addition, automated backups are crucial to a cost-effective solution. Finally, you need to develop a strong and well-documented testing methodology for your backup procedures to ensure that changes to your network and UNIX systems don't compromise your backups' reliability.

4

Change Management

IT managers need methods for managing the changes that an evolving organization requires. Systems periodically require updates, configuration changes, and software deployments; effectively managing these changes is crucial to network and system productivity. Although most UNIX administrators automate repetitive processes, administrators often handle change management manually. This practice is unwise not only because it's time-consuming but also because you can easily make incorrect changes that are difficult to reverse. Bad change management decisions can cause problems from lost Internet access in a small company to substantial revenue loss during extended downtime in a large corporation.

An important part of change management is ensuring that your OS remains reliable and manageable; if your OS is unavailable, you can't justify the cost of investing in UNIX. Regardless of how well IBM's DB2 can run on AIX, if a systems administrator makes a poor configuration choice then the system will probably experience downtime (e.g., if a systems administrator takes an AIX disk volume offline for maintenance and fails to bring it back up when done). An important goal of change management is to eliminate or reduce downtime for servers and services.

In this chapter I discuss change management's core concepts, including how change management affects UNIX systems managers' adherence to best practices. In addition to defining change management, I explain how to apply change management to your organization's UNIX infrastructure. This chapter will help you follow current standards and best practices for implementing change management.

Change Management Philosophy

Many resources (e.g., books, articles, seminars) focus on the technologies behind effective change management in both UNIX environments and entire organizations. Although these resources can help you implement technologies to assist with change management, the resources often miss the real point of change management. Change management is simply a philosophy for managing systems and configurations.

Although automating software updates and ensuring that users' change requests receive prompt attention are reasonable expectations, change management's main goal is to provide consistency and manageability. Consistency involves more than just ensuring that no differences exist between configuration files on separate systems. Consistency means effectively managing the workflow (i.e., how change requests are made, executed, completed, and documented). The technology you purchase to manage change management can work only if the technology implements your change management goals.

An important element in considering change and configuration management for UNIX systems managers is that many UNIX environments are heterogeneous (i.e., multiple versions and flavors of Linux and UNIX are used). Having a diverse UNIX network can increase the difficulty of managing change and configurations because how, where, and why changes are made vary between systems (e.g., AIX uses different tools and configuration files to manage password aging than does Linux). The systems you plan to support determine the tools you use (or how much time you must spend to customize tools for your network). As I mentioned in Chapter 1, this book stresses the importance of having a common management infrastructure built on a heterogeneous UNIX network. Companies can rarely consolidate to a uniform Linux distribution or UNIX flavor; an alternative approach is to provide a common foundation for UNIX systems management and to build or purchase tools that implement your changes for each UNIX system you run. (Cfengine is a good example of a tool that can provide a common management structure for all your Linux and UNIX systems.)

Note: Linux, which is now common in enterprise networks, plays a large role in how you manage your systems. Variations between Linux distributions can be as significant as variations between UNIX flavors (e.g., Solaris and Hewlett-Packard's—HP's—HP-UX). Although Linux distributions have many commonalities, some core systems administration functions (e.g., managing installed software packages) differ considerably. Your management tools must be able to neutralize the differences between Linux distributions.

Change Management Goals

Change management applies to a variety of situations. For example, most software development companies use revision control software to practice basic change management. For these companies, the ability to document changes lets the company quickly locate problem programmers or more easily roll back from a failed change in an algorithm. Change management is similar for UNIX systems managers, although the focus is on managing large, complex systems and environments rather than on managing source code.

Note: I was a software development consultant for a client creating a sophisticated network appliance. The company needed to manage a large set of custom software and configure and customize the underlying OS. When I started the project, the company hadn't implemented revision control for its software or configuration files. Over the course of several months we moved all the source code and configuration files into Concurrent Versions System (CVS, which I discuss later in this chapter). We imported initial versions of source

code, then applied the latest version so we had a record of changes. Having the source code and configuration files in CVS let the development team easily track changes. Developers could see each change, including who made the change and the reason for the change. This information made development, peer review, and collaboration easier.

When considering change and configuration management, UNIX systems managers must be aware of a system's ideal state and the system's actual state. Understanding the difference between an actual system and an ideal system is crucial to change management and configuration management.

Systems that vary from the ideal state are called divergent. (The concepts of convergent and divergent systems are important to the cfengine configuration management tool, which I discuss later in this chapter). All systems naturally diverge from their ideal state over time. For example, in an emergency a systems administrator might make a quick fix to a Linux server's `/etc/fstab` to update which disk volumes are mounted, or change a database application's startup script to adjust the default process limits (e.g., `ulimit`) that the OS places on the database process. Small changes add up over time, until eventually the system diverges so much that manually reconciling the differences between the ideal system state and the current system state is difficult. Conversely, as a system returns to its ideal state, it converges. When a system converges, you can more easily reconcile the differences between the system's actual state and ideal state.

Ultimately, one of change management and configuration management's primary goals is to increase convergence. Increasing convergence in a UNIX environment involves managing configurations and software.

Configurations

Configuration management is a component of change management. Although some UNIX systems now store configuration information in traditional databases (e.g., AIX), UNIX systems conventionally store their configurations in human-readable configuration files. These files are typically in locations such as `/etc`, but many applications maintain their configuration in other directories (e.g., `/usr/local/apache/conf` for Apache, `/opt/websphere` for IBM's WebSphere). Thus, configuration management in UNIX networks primarily means how you manage a system's configuration files. These files dictate the behavior of the system on which they reside. (I discuss managing UNIX system configuration in more detail later in this chapter.)

Software

Change management can play a pivotal role in software deployment (i.e., how software is rolled out to large groups of systems). Most companies that maintain several servers deploy specific sets of software across many of their servers. This software can include mail software for mail clusters, Apache for Web servers, Oracle for database servers, and text editors for software developers.

Maintaining consistent software deployment across large sets of servers is even more difficult than maintaining consistent and documented OS configurations. You might need to deploy various software packages across several UNIX flavors, and each flavor might require specific system library versions, add-on software, or configurations.

Note: Linux further complicates software management because of the differences in software management between Linux distributions. Most Linux distributions support the Red Hat Package Manager (RPM) format, which Red Hat introduced to manage software deployments for the company's own Linux distribution. Although standardizing to RPM for software management on Linux servers isn't necessary, you need to pick a standard format and consistently use that format across your business's Linux distributions. Doing so reduces the overhead in managing and testing software before deployment and improves how your tools implement software installation.

Auditing

Pushing configurations and software to servers is important, but you also need to be able to confirm system states after changes. Security policies often require security or audit departments to regularly review systems, server software, and end-user access to ensure consistent, reliable, and secure settings and defaults. IT departments must discover which changes diverge from a system's ideal state and determine why the divergence occurs.

Change management is important in auditing changes across your enterprise. Many change management software packages not only help you define and deploy changes across the network but also let you review why, when, and how the changes are accomplished. This ability aids in long-term trending of systems administrator and management effectiveness and security, as well as the change management system's effectiveness.

Reasons for Change Management

You must first understand the reasons for using change management if you hope to convince your organization to implement change management. Although organizations seek cost-effective solutions, the status quo can determine which solutions are implemented. Managers often resist change management because they believe change management will add to their company's bureaucracy and prevent the company from reacting quickly when changes to UNIX systems are necessary.

Systems administrators often resist implementing a change management procedure because they mistakenly believe the new procedures will make their jobs more difficult. This resistance is actually an indicator that you need a set of change management policies and procedures. In sites without a formalized structure, systems administrators often make changes without researching and testing the changes. This practice is dangerous in an enterprise network, in which you measure UNIX server uptime in months rather than weeks or days. Ad hoc changes to UNIX servers can cause long-term problems, including lack of documentation and loss of changes after reboot. Proper change management procedures have a direct and quantifiable positive effect on how you manage your UNIX servers.

Better Use of Staff

Staff costs are often a company's largest expense. Highly skilled and experienced Linux and UNIX administrators are expensive to employ and train. Computers can accomplish

some tasks, but computers can't compete with humans' ability to reason, provide insight, and act on instinct to define problems and offer solutions. However, companies still underutilize their staff. Management sometimes spends a large portion of the personnel budget to maintain staff to implement changes rather than to manage change. Software is better suited than humans are to implement changes, a task that requires the ability to repeatedly perform redundant and complex tasks without failure.

You need to use junior staff to assist software and hardware change management systems with daily tasks. Your senior systems administrators and managers are better suited for managing the short- and long-term trends and changes necessary to make your organization effective. Experts should manage change rather than devices. Asking senior UNIX staff to manage devices and servers wastes their time and your money.

Note: Using junior staff to help test and implement changes helps the staff gain on-the-job UNIX network experience. Testing changes is also a good way to train junior and senior systems administrators on new UNIX systems in your organization (e.g., Linux).

Risks and Rewards

Managing change is often more about managing risk than ensuring that systems administrators have an easy job. All changes to your UNIX infrastructure have an associated risk; change and configuration management help mitigate those risks. Although change management saves money on staff costs, other reasons for using it are equally or more important. Implementing change management can increase security, decrease downtime, and reduce the potential for human error.

Increased Security

In some settings, downtime is a preferred state if a system or network isn't properly configured. For example, in many military environments, the preference is for a system to fail completely rather than partially. Suppose that an administrator updates a DNS server without following proper change management procedures, pointing a military domain's MX record to an unsecured, off-base server. This configuration error won't result in network downtime, but sensitive information is at risk. Using the proper change management tools and procedures will help you quickly find and solve such a problem.

Note: Because Linux and UNIX are so file-centric, you can use change management tools not only to implement and track changes but also to quickly detect unauthorized changes. This ability gives you Tripwire-like capabilities in one change management package, thus reducing the amount of software you need to train your administrators to use. (Tripwire—<http://www.tripwire.com>— detects files changes on UNIX systems.)

Increased Uptime

Change management redirects a staff's focus from implementation to review and documentation. Engineers and systems managers review proposed changes not only for implementation details but also for implementation effects. Change management lets staff devote more time to review because they spend less time deploying changes.

UNIX has a long history of high uptime, and Linux is quickly earning a similar reputation. Properly testing and reviewing changes before implementation ensures that servers don't have unnecessary downtime.

Improved Documentation

One of change management's most important benefits is better documentation. Strong change management policies and procedures focus on change request documentation, change implementation, and post-implementation effects. Over time this documentation further reduces the amount of time necessary to review and implement changes.

Most sites already maintain systems administrator logs (i.e., written or electronic logs of systems administrators' actions). Letting the change management system provide more accurate information about which files were changed, including when and how, complements this documentation to let you more easily reproduce changes as you reinstall servers or move to new hardware.

Note: A simple trick for encouraging systems administrators to log their actions on UNIX systems is to provide a script on each server that lets administrators quickly log their changes. Although this method doesn't replace a proper change management system, the procedure eases administrators into using a more regimented and documented method. The script you use doesn't need to be complicated. For example, you can simply send the administrator's log message to a file on the local server, as the following code shows.

```
#!/bin/sh
PATH=/bin:/usr/bin
echo "`date` $1" >> /var/log/change.log
```

To call the script, enter the following code (assuming the script is named `log_change`).

```
# log_change "Updated /etc/fstab to bring up the new
disk that we installed."
```

Be sure to back up the file in case the server goes down. Consider writing the script to keep logs locally and send logs to a centralized database so that you can easily create management reports on demand.

Reasons for Avoiding Change Management

Few if any reasonable objections to change management exist. However, people often resist change and therefore might object to a change management movement in an organization.

Red Tape

Many people object to change management because they assume that change management procedures involve lengthy review cycles and require copious amounts of information to justify changes. Change management certainly introduces some bureaucracy. Large organizations with large networks to manage might require lengthy review cycles, but small companies can often implement change management procedures for fast processing of change requests. In either case, the reduced time necessary to

implement and troubleshoot changes in a properly defined change management procedure offsets the increased time necessary for the review stage.

The Status Quo

Maintaining the status quo is often desirable—If it ain't broke, don't fix it. However, an organization without a formally defined change management policy might be losing substantial resources maintaining a system that's difficult to use. Ask yourself whether your current unofficial change management system at least ensures that you maintain proper documentation. (UNIX servers are notoriously flexible and configurable, and the files configuring your UNIX servers and applications change over time. How many of your file changes are undocumented?)

Most reasons for not formalizing change management are invalid. Change management helps you manage your UNIX systems, ensuring that both your servers and systems administrators are effectively used. Change management enforces a consistent framework from which administrators work and ensures a minimum level of compliance for servers.

Note: Linux systems administrators often have less experience than UNIX systems administrators, especially in small companies. Linux administrators therefore make more changes, and with less documentation, than UNIX administrators. You need to ensure that your Linux systems administrators are trained in administration and in maintaining properly configured and documented systems.

Define a Process

At its simplest, change management is a codification of the procedures you use to request and implement changes on your network. Although I discuss change management within a UNIX infrastructure, change management applies to all the platforms, networks, and communications a company uses.

The first task in implementing change management is to define a simple change management procedure. Figure 1 shows the five main change management components in a UNIX network: change request, approval, scheduling, execution, and documentation.

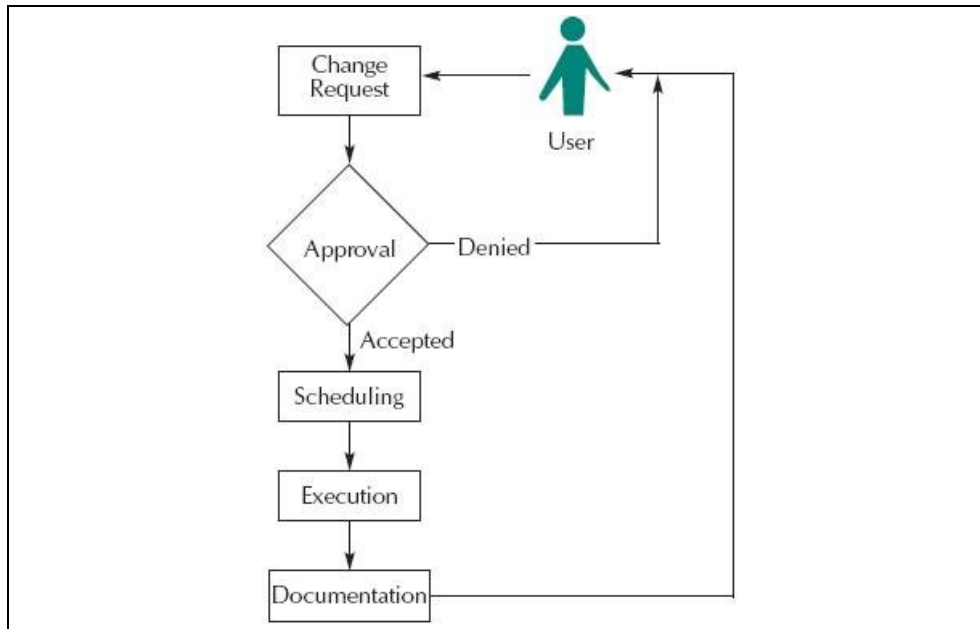


Figure 4-22. Change Management Components

In the change request stage, a user requests a change. For example, a database manager might request additional storage space for databases. The manager doesn't satisfy this request himself or herself. Instead, he or she submits a request to the management team. The team then reviews the request, determines whether the request is valid, and accepts the request. Next, the team executes the change, documents the change made and reasons for the change, and informs the user of the change.

Note: I use the word team in explaining the change management process because a well-designed and implemented change management procedure has more than one UNIX expert who reviews and approves requests (particularly requests that greatly affect your systems). Letting only one UNIX guru approve and deny requests is a mistake.

This change management process works well for a small network. But if you have a large UNIX management team that manages hundreds of systems, you might have trouble reviewing and approving change requests in a timely manner. In addition, ensuring that two people aren't working on the same problem is difficult.

Refine the Process

After you define a change management process, you need to refine it. Determine how your existing standards affect the change management process and how the process affects your existing standards. In the following sections I discuss several important issues for you to consider, including suggestions for resolving those issues.

User Base

To refine your change management process, you need to identify whom your procedures affect and determine how to best apply a new change management procedure for those people. For our purposes, I discuss how change management affects users, managers, and systems administrators.

Users include general users, power users, and remote users. These people rely on your UNIX systems daily for tasks such as checking email, submitting reports, and tracking orders. Users often respond the fastest to a problem. That is, if your monitoring tools don't detect a problem, your users probably will.

Managers are a company's upper-level decision makers. Although managers are also users, managers have much more power than users to influence decision and prioritization processes. You need to balance managers' influence to best use their positive effect and minimize their negative influence on your processes.

Systems administrators often take the easiest route to solve simple problems. This practice can negatively affect your UNIX infrastructure's performance, security, and reliability. For example, a systems administrator who notices that a process is hung might simply fix a configuration problem and restart the process. If that administrator then leaves the company, you have no record of why the change was required and how it was implemented. The problem might then reoccur. A better procedure for the systems administrator to follow would be to log a trouble ticket, perform troubleshooting, document the intended change, implement the change, then restart the process. (Note that if the application is immediately required, the administrator might need to restart the process before troubleshooting the application and documenting the change.) Requiring administrators to use this process documents changed system states and adds to your company's knowledge base of problems and their solutions. Getting systems administrators to accept your change management procedures is crucial. Otherwise, you might establish an excellent change management procedure that nobody uses.

You might want to consider different groupings regarding whom your change management procedures affect. For example, you might combine users and managers into one group. Regardless of how you categorize people, you need to analyze how your user base works together and across political divisions and determine how those dynamics affect your procedures.

Systems, Devices, and Networks

Systems managers and administrators manage systems and often network devices that connect systems together and to other networks. When you establish change management procedures, you need to define the set of systems that the policy will manage.

A typical enterprise network has hundreds of different types of network devices, OSs, and printers to manage. This book focuses on Linux and UNIX systems, which considerably narrows your management requirements. In a Linux or UNIX environment you need to manage applications and their configurations, UNIX user groups, password aging, and even the kernels that run your servers.

Implementation

When you establish a new change management procedure, you need to determine how to best implement the procedure in your environment. Changing your procedures and policies often undermines your current processes' consistency, especially if you fail to educate key users about the new procedures and policies. Implementing change management in an organization involves more than just establishing a set of guidelines; you must also educate the people the new guidelines will affect.

Procedure

The change management components I discussed earlier are a good starting point but don't fully encompass the steps in change management. Figure 2 shows change management as a 7-step workflow. These steps include:

1. Change request
2. Review and approve
3. Assign
4. Research and test
5. Schedule and execute
6. Document
7. Close change ticket

This workflow process works well in large environments because it lets you push several change requests through the pipeline simultaneously. You can assign several people to each step in the process as necessary.

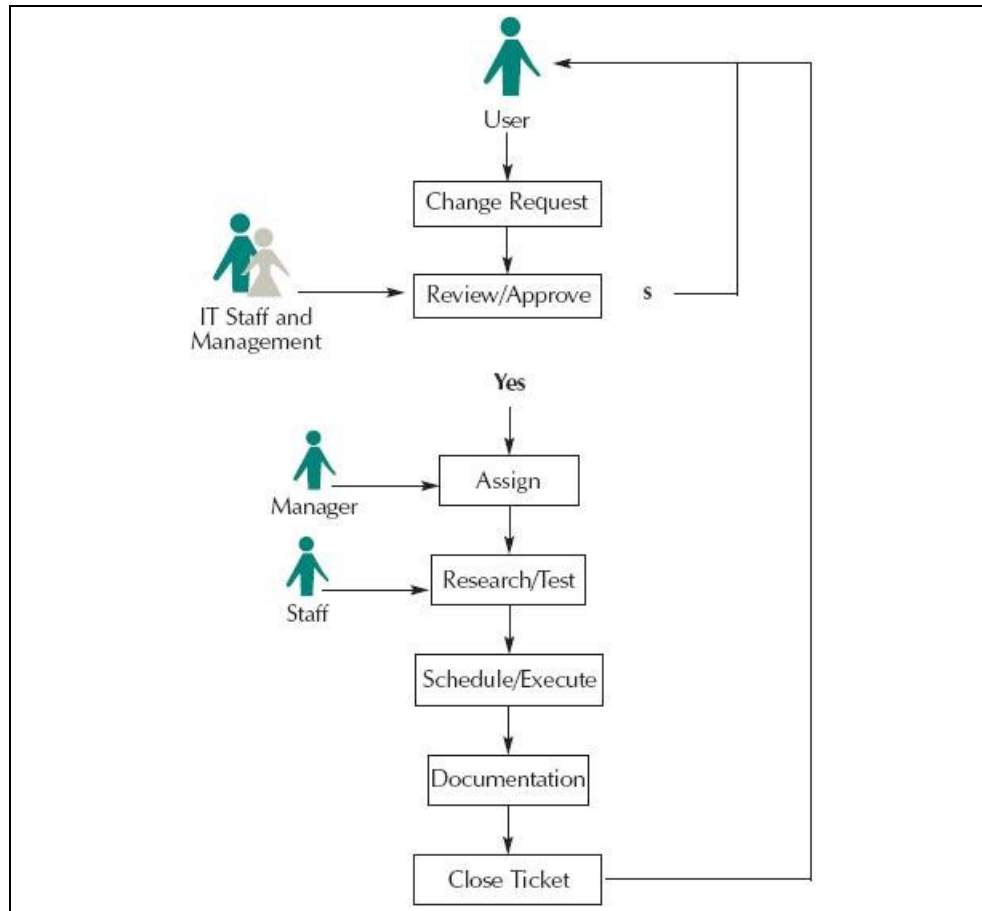


Figure 4-23. Change Management Workflow Process

Step 1: Change Request

The first step is for a user to make a change request. Who makes the change request is important because this information can affect the priority you later assign to the request. The complexity of change requests varies within and between user groups. An end user might request an increased home directory quota, whereas a systems administrator might request an entire new drive allocated to the home file system.

Assuming you have an easy to use and understand ticketing system to manage your change requests, which I discuss in more detail later in this chapter, users can make these requests without your management team's help. You need a Web-based interface for users to issue requests quickly, easily, and directly.

Step 2: Review and Approve

The next step is for someone to review and approve (or reject) a submitted change request. Who approves and rejects change requests can cause problems in an organization. If the person in charge of this step rejects a change request, future problems might arise. (For example, suppose a mail server administrator's request for more storage space is rejected. The mail spool file system might then become full, stopping mail from

coming into the company.) Conversely, if the person accepts a request that he or she should reject, valuable systems administrator resources will be wasted.

Deciding which requests to accept or reject requires experience and a firm understanding of the company's goals and priorities. Some administrators reject few requests but assign a low priority to requests that don't significantly affect the company's business (e.g., a request for a faster video card for a secretary who uses a UNIX workstation and WordPerfect).

In this stage you might want to assign administrators roles for monitoring requests. For example, Help desk personnel can assist in monitoring general users' change requests, whereas fellow systems administrators can monitor other systems managers' and administrators' requests. This method ensures that the right people manage incoming change requests. If your organization is small you might not need such a complex split in roles; if your organization is large you might want to further split the roles.

Step 3: Assign

The next step is to assign responsibility for the change request to a systems administrator or team of systems administrators. You need to assign responsibility as soon as possible so that one person or team can process the change request and ensure consistency.

Deciding who to assign a ticket to is usually simple. IT staff subgroups typically manage specific UNIX infrastructure components; if a change request affects a particular IT team, assign the ticket to that team. For example, you'd give the Storage Management group a systems administrator's change request for more storage capacity on an Oracle server.

Rather than assigning relevant change requests to a specific administration group, you might instead prefer to rotate requests throughout your IT department. This method gives your entire team a broad set of knowledge. If a key team member quits or transfers out of your department, other team members can easily assume that member's workload. A drawback to spreading change requests throughout your staff is that an administrator might have difficulty becoming an expert at a specific set of tasks.

Your organization's requirements and internal politics will determine how you assign change request workloads. In my experience, cross training your staff is more important than having a few people who excel at particular tasks. However, don't overemphasize cross training; despite differences between UNIX flavors and Linux distributions, UNIX administrators can quickly learn new UNIX OSs. For example, although AIX and Solaris have different management tools, a Solaris administrator who has access to AIX documentation and the Internet can attain a junior level of AIX expertise in only a few days. Although you need to include cross training in your administrators' education and management, your administrators also need time to better learn the UNIX systems they manage daily.

Step 4: Research and Test

After you assign a change request, the responsible systems administrator needs to research the request. The administrator must decide how to implement the change, which involves determining whether the change was implemented in the past and the change's results if so. For the administrator to effectively and efficiently complete this task, you must have a solid change request tracking system and a documentation process for actions taken on change requests.

After the administrator researches the change request, he or she must try out the change in a test environment and document any problems that result. Even if the administrator has previous documentation that alerts him or her to possible problems, new problems can occur. Testing changes minimizes potential problems on your production network.

Note: IT departments often maintain several test servers to test changes before implementing them in a production network. A test network can consist of only a few to several hundred servers. VMware (<http://www.vmware.com>) is software that lets you concurrently run several virtual servers on one computer. This software can dramatically reduce the cost of maintaining a test network. VMware also lets you easily restore a server to its pretest configuration.

Step 5: Schedule and Execute

After research and testing, you need to schedule the change, then execute the change request. When scheduling a change you need to consider what systems and users the change will affect. This information helps you determine the best time to make the change. For example, if the change will bring down a network interface for several minutes you might want to avoid scheduling the change during normal work hours or during the backup windows.

Next, execute the change. A checklist for executing a change on a production network is helpful. This checklist contains the steps necessary to implement the change, as well as instructions for backing out of the change in case a problem occurs that you can't immediately solve. Even after testing, a change can cause unforeseen problems. Your back-out plan must be available for several days so that you can quickly revert to your original configuration if problems occur later.

Note: Backing out of UNIX changes can be simple if you properly document your changes. Because UNIX stores most configuration parameters in files, in most cases you can comment out the old configuration and include the new configuration. Using this method to make changes lets you comment out the new changes and uncomment the original configuration to revert to the old setting. The problem that usually occurs is the downtime while the bad configuration is active. You need to carefully test and document your changes to avoid excessive downtime.

Step 6: Document

Next, the systems administrator must document the system changes. Depending on the change's complexity and its effect on other components, the administrator might work with a systems manager to document the change's short- and long-term effects on the UNIX infrastructure. For example, significantly changing a DNS server's organization and configuration requires more detailed documentation than simply adding a user to a UNIX server.

Note: Documentation that isn't readily available is useless. Systems administrators need easy access to documentation for changes made to systems, even if the ticketing and change management applications aren't available on the network (e.g., if the network crashes). To give administrators quick access to documentation, you can publish changes

on an internal Web server or on a backup documentation server that administrators can access even during emergencies.

Step 7: Close Change Ticket

Several days after you make a change, you can close the change request ticket. Keeping a ticket open for an extended time helps you determine whether a change might have caused an increase in problems or Help desks requests. If a change causes a problem, you can assign the problem to the change's ticket and include documentation about what occurred and why. This extended documentation can provide information to speed subsequent research and testing on the change.

When you close a change request ticket, you also need to notify the person who requested the change that you've implemented the change. Although you should have tested the change before implementing it, the person who requested the change is the best judge of whether the change was effective.

Note: Even after you close a change request ticket, you aren't done with the job. You need to consider how the change might affect your disaster recovery procedures. For example, if the change will affect system reinstallation, you need to fully review the change and update your disaster recovery documentation accordingly. Because you need to integrate this information into your disaster recovery documentation and procedures, you must consider a change's long-term effects during the review stage rather than after you make the change.

UNIX Change Management Tools

Although change management is a philosophy and process rather than a technology, using technology to make change management more convenient is helpful. Software helps you implement change management and guide your users, administrators, and systems managers in managing both servers and change management processes.

Request Tracking

One of the most difficult change management tasks is tracking submitted change requests. People often make change requests over the telephone, in the hallway, and during meetings. To reduce these undocumented requests, you can implement an electronic-based system that lets users easily make change requests and gives administrators the functionality and history they need.

Templates

Templates help ensure consistency in request submissions. Users often give a general overview of the change they're requesting, without providing details. Although administrators have more technical knowledge than users, only the user requesting a change knows exactly what he or she wants changed and why. In many cases, the reasons for a requested change greatly influence how you process the change request.

Keep in mind that a user might confuse the reason for a change with the change itself. For example, a user might want an application to run faster and therefore request more RAM on the server. The change request is adding RAM to the server, whereas the reason for

the change is faster application execution. The systems administrator might determine that the server doesn't need more RAM but that the user needs a faster network connection to the server. Sometimes a user knows that a change is necessary but not the best way to make the change.

Figure 3 shows the type of information a change request template might include. The example form has fields for the user's contact information (including department), change requested, timeline needed, and problem urgency. Each field is important. The department is necessary so that you can contact the user's manager if you need to follow up on the reason for the change or obtain authorization for the request. The timeline and urgency help you schedule the change implementation. Although a change request such as increasing a user's mailbox size might seem important, if the user doesn't need the change immediately you can schedule the change with other changes to consolidate tasks.

Contact Information
Name:
Phone:
Email:
Department:
Manager:
Change Request
Change Requested:
Reason for the Change:
Date Needed By:
Urgency: Low / Medium / High
Systems Affected:
Back-out Plan:

Figure 4-24. Example Change Request Form

Workflow Enforcement

Getting users to follow your change management procedures makes the process more efficient. You can accomplish this goal informally through repeated education or formally through the change request tracking system. Most advanced ticket tracking systems, such as Best Practical Solutions' Request Tracker (RT—<http://www.bestpractical.com/rt>) and BMC Software's Remedy (<http://www.remedy.com>), can implement your workflow within the ticketing system (e.g., which changes require approval, who approves changes).

Document Request Resolution

In addition to documenting and tracking change requests, you need to document each request's resolution. If a request is denied, you need to record the reason for the denial. Over time you can see what kind of change requests users make and why requests are accepted or rejected. When you determine why certain requests are rejected, you can educate users about appropriate requests. In addition, having a record of requests can help you identify patterns in the kinds of problems users report and the changes they request. You can use this information to proactively make changes (e.g., granting users more disk space when they transfer to a department that requires them to maintain more files) rather than wait for users to alert you to problems.

Ticket Systems

Several ticket-tracking systems exist, including Remedy and RT. RT is an excellent general-purpose system for UNIX environments. This program runs on Linux, UNIX, and FreeBSD and integrates well with native mail software for ease of use in accepting and responding to tickets. RT isn't written specifically as change management software, but you can customize the software to track change requests and resolutions.

Tickets systems alone don't comprise a comprehensive change management toolset, but they provide a foundation for tracking requests and resolutions. Without a well-implemented change request tracking system, you'll have difficulty ascertaining which requests have higher priority, finding the problem that caused a request, and determining whether a request was made previously.

Change Implementation

You must consider several issues when you implement a change management tool. These issues include mixed environments, vendor tools, extensibility, restoration, and auditing.

Mixed Environments

In deciding how to best define and implement your change management procedures, consider how the procedures affect not only the UNIX infrastructure but also other systems and the network. Most modern enterprise environments are heterogeneous—that is, they contain a large number of OSs. Work toward a unified change management procedure that's applicable across managed systems and devices. A unified change management infrastructure lets you better consolidate your systems management staff and the tools they use, thus increasing the benefits of staff training.

The term *mixed environment* also applies within UNIX because UNIX has many flavors. For example, an environment that includes Linux, Solaris, and AIX is a mixed UNIX environment. Each UNIX flavor has a set of management tools; you must use the tools that operate within a varied environment. In other words, use tools that manage across UNIX flavors, rather than tools that manage only one UNIX flavor or Linux distribution.

Because most enterprise environments are mixed, systems managers must be wary of using internally developed change management toolsets. Tools created internally are often designed for specific UNIX systems. As you expand the types of systems you manage, these tools can become ineffective. A good example is increasing the use of Linux on a UNIX network. Linux has several distributions, much like UNIX's many flavors. If you design a tool for a specific UNIX system, the tool might become ineffective when you begin using different Linux distributions (e.g., Red Hat, Debian).

Vendor Tools

Arguments exist both for and against using vendor tools in change management. Many vendors provide informative and usable change management tools for devices the vendor supports (e.g., Check Point Software provides software to consistently manage multiple FireWall-1 firewalls).

The main problem with vendor tools is flexibility. You can rarely expand such tools beyond the vendor devices they were built around. In a large environment, using a vendor's tool that works only with the vendor's products might exclude large groups of devices and systems from your change management solution. You'd need to manually configure such systems, thus defeating change management's purpose.

Another problem with using vendor tools for specific devices in change management is the time you spend educating your systems administrators to use the tools. If a systems administrator learns to use a specific tool to manage a small but important subset of your systems (e.g., AIX), that administrator's skills aren't immediately applicable to other systems. With a more comprehensive set of change management tools (i.e., tools that are applicable to multiple UNIX flavors), administrators can more readily use their skills as different UNIX flavors and applications come under their control.

Extensibility

Your change management software needs to be extensible, to give your procedure flexibility. The systems you manage and how you manage those systems change over time. Your change management tools must be adaptable across systems. If your Linux- and HP-UX-only environment expands to include Solaris, your change management infrastructure must be able to seamlessly support Solaris.

One way to judge a set of software's extensibility is to determine whether the software can define a configuration for a server that isn't specific to an OS. For example, a high-level language such as cfengine lets you define configuration parameters for a DNS server and allow the change management tools' agent to decide how to best implement the changes based on how to configure the agent for each server and UNIX flavor.

Note: Unlike most other OSs, in Linux and UNIX your change management software's extensibility is a required feature rather than just a nice benefit. Variations between UNIX flavors and Linux distributions are so wide that a change management tool must be able to adapt to changes across different UNIX versions.

Restoration

A good change management tool supports system rollback or at least restoration to the Last Known Good configuration. Despite testing, some of the changes you implement will negatively affect your systems. Your change management tool needs to be able to reverse changes rather than you having to manually change systems back to their original states.

Being able to restore the Last Known Good configuration is more important than many systems managers realize. As systems evolve, their configurations become more complex. If you make a bad change, rebuilding the original configuration from scratch can be difficult. Restoring the Last Known Good configuration can save valuable time.

Auditing

You need to be able to audit changes made to systems. A good change management tool can apply changes and detect when unauthorized changes are made to the system. Change management tools are effective auditing tools because they can detect variances in a system's configuration from the ideal system state. In addition, some change management tools can detect unauthorized system changes and therefore act as a warning system.

Change Management Software

No single change management solution exists for Linux and UNIX systems. Large companies typically integrate various features from different software to offer a full range

of services (e.g., ticket tracking, logging of server changes). A good approach is to use a strong change request and trouble ticket tracking system, such as Remedy or RT, along with configuration management software, such as CVS or cfengine.

In the following section I discuss some open-source software that you can use for configuration management. This software works in many environments; determine your requirements before you decide which software to use.

RCS and CVS

Many systems administrators use UNIX's Revision Control System (RCS) software to manage configuration files. A software developer or systems administrator can use RCS to check in a file before making a modification. The following code shows an example of using RCS to check in a server's `/etc/fstab` file, possibly before editing the file.

```
$ ci 1 /etc/fstab
```

When the administrator checks in the file, he or she logs the reason for any changes; RCS stores those changes in a history file. The most recent version of the `/etc/fstab` file in my example is left in the `/etc` directory; a copy is stored in `/etc/fstab,v`. This approach lets you easily find differences in files in case of a mistake or if you need to assess past changes. RCS works well for small networks or single servers. However, the software doesn't scale well in larger environments. One problem is that the history file must be in the same directory as the original file or in a subdirectory named RCS. Thus, to determine which files were modified you need to search for all files ending in `,v` or for the RCS subdirectory. Systems administration and auditing are difficult because you can't quickly determine how far a system diverged from the ideal state.

CVS (<http://www.cvshome.org>) was designed to replace RCS. CVS is a multiuser, networked version of RCS, as Figure 4 shows. Although CVS doesn't directly assist with change management, the software can aid significantly in configuration management. CVS stores all files in a central repository on the network. CVS uses RCS-like files. A useful feature is the ability for multiple users to check out and edit a file; when users submit file modifications, CVS attempts to reconcile the differences.

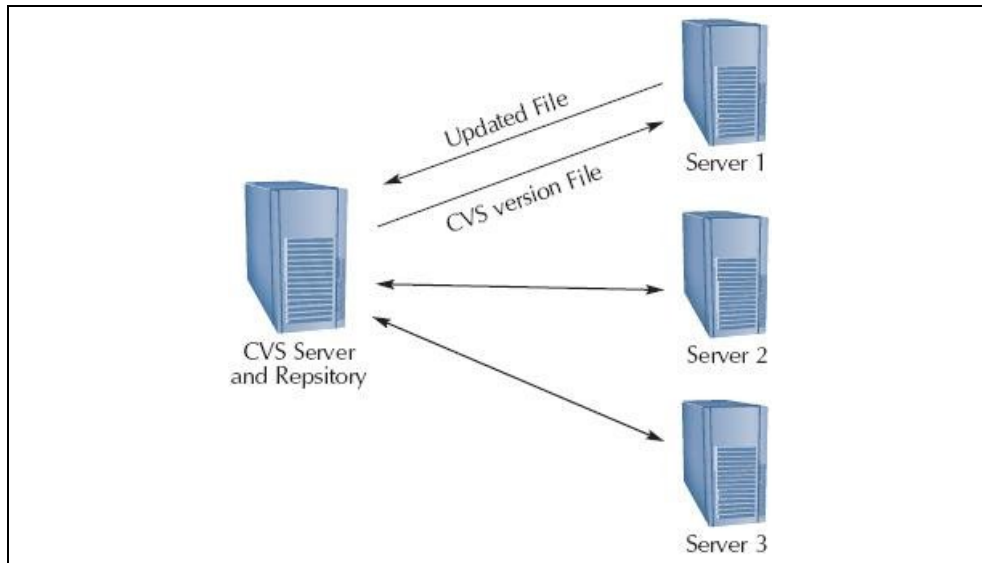


Figure 4-25. The CVS Network Model

Most UNIX environments need to be familiar with RCS and CVS because of the software's wide use in configuration management. CVS can be pivotal in managing configuration files if you use the software in conjunction with a strong change management procedure.

Cfengine

The configuration engine (cfengine—<http://www.iu.hio.no/cfengine>) goes a step beyond CVS. Cfengine is a language-based system that uses certain software's (e.g., CVS's) best features to aid in configuration management. Figure 5 shows cfengine's role in maintaining a UNIX network.

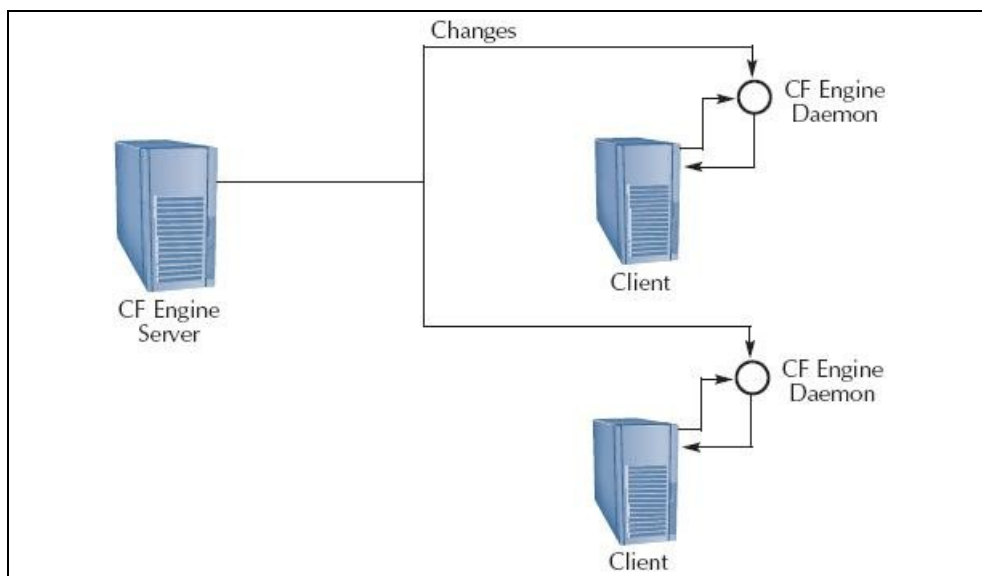


Figure 4-26. Using cfengine to Maintain a UNIX Network

Cfengine lets you consolidate configurations to a central repository, then automate system convergence against that configuration data. You establish a set of definition files that apply to your systems. Cfengine automatically changes systems that diverge from these definitions so that they begin to converge. One of cfengine's useful features is that you can define a class for each server to belong to. Rather than maintaining definitions for each system, you can maintain definitions for each class of systems. As you install new servers, you assign the servers to a class; cfengine brings the servers into a state of convergence. This automation makes change management and server installations easier.

Conclusion

This chapter explores change management and configuration management. Change management is a philosophy rather than a specific technology. Best practices for applying change management involve separating each stage into a specific, identifiable component of the entire process. Configuration management is part of the larger change management process. You can use a commercial package or a widely used tool such as CVS or cfengine to implement configuration management.

5

Performance Management

Performance management and tuning are complicated. Many factors affect what systems managers and consultants consider good or bad performance, and many other factors affect the end-user experience.

In general, latency and throughput affect performance. Many systems administrators, managers, and users confuse latency and throughput. Latency is the length of time necessary for a measurable value to reach a user (e.g., a byte of data crossing a network—a common method for measuring a TCP/IP network’s latency is to ping a remote host). Throughput is a measure of how much data can transfer effectively at one time. A trade-off often exists between latency and throughput; your responsibility as a systems manager is to determine which takes precedence.

Latency greatly affects the end-user experience. If someone using an X Window System application clicks a button, a measurable amount of time passes before the X server transmits the click event to the X client and the X client updates the X server’s screen. This is an example of latency.

Bandwidth determines throughput. As you increase how effectively you use bandwidth, you also affect latency. For example, consider a network connection between two offices. If you enlarge the packet size to more effectively use bandwidth (thus reducing protocol overhead), you consume more of the pipe for longer periods, thus increasing effective latency. You can transfer more data in a time interval (increased throughput) but with a corresponding decrease in response time (increased latency). The reverse situation can also occur: If you decrease the packet size to decrease latency, you subsequently use the network less efficiently.

No simple answer exists for the question of whether latency or throughput is more important and where your focus should lie. Each application and user population has unique requirements. A user accessing an interactive UNIX application is more concerned with latency, whereas a backup operator is more concerned with throughput. This chapter discusses issues that affect both latency and throughput.

The most valuable unit of measurement is performance. Systems administrators often focus on disk I/O or on ensuring that their UNIX servers are all running on Gigabit Ethernet. They mistakenly neglect performance management and tuning.

In this chapter I discuss best practices and rules of thumb for performance management and tuning. Although systems managers must effectively manage several UNIX systems' performance, they first need to learn how to manage just one system. Therefore I focus on understanding and tuning one system, then expand the information to include networkwide UNIX performance management.

This chapter includes several sections called Real-World Performance Tuning. For more information about real-world tuning, see my *Sys Admin Magazine* article "Linux Kernel Tuning Using System Control" (November 2003), UNIX-specific magazines and books, and your UNIX flavor's manual.

I often refer to the Apache Tomcat J2EE server as an example to demonstrate when and how to tune systems and to show why performance management is important. Tomcat is a Web server that runs server-side servlets and JavaServer Pages (JSP); Tomcat essentially runs Java applications on the server and returns the results to the client. You can download Tomcat from the Apache Jakarta Project Web site (<http://jakarta.apache.org/tomcat/index.html>). For more information about Java performance tuning, see the Java Performance Tuning Web site (<http://www.javaperformancetuning.com/tools/jamon>).

Obtaining Performance Baselines

One of the first steps in establishing a performance management strategy is to determine how your systems should behave during a normal load. An organization that hasn't collected performance baselines for its servers has no performance management strategy, regardless of the company's performance troubleshooting tools and technologies. A difference exists between administering a server and managing a server. Management focuses on the short- and long-term strategies necessary to maximize your UNIX assets; management requires long-term historical data for new decisions (e.g., whether a server can handle an additional load).

Your first job is to measure your server's performance. You need to obtain at least two measurements: the UNIX server with no load, and the server with a normal load. Measuring a UNIX server with no load is simple, but measuring a normal load can be difficult. First, what constitutes normal? And if you don't have a baseline, how do you know whether you're measuring a normal load? One way to define normal is to run a system with a projected number of users (real people or remote Web servers), then monitor the system's performance.

Performance Monitoring

An integral part of performance management is daily server and application performance monitoring, including monitoring UNIX servers' low-level details (e.g., disk and network I/O, CPU utilization, memory consumption, virtual memory use). This data is the most often measured, but you must go beyond simple data gathering. Systems managers must ensure that end users have timely access to the applications and data they need.

One of performance monitoring's main principles is to limit your effect on the system you're monitoring. That is, your monitoring shouldn't affect the data you collect. The tools I discuss in this chapter (e.g., `sar`, `vmstat`, `iostat`) don't have a large effect on the monitored system. Other tools (e.g., `top`) affect lightly loaded servers. Fully test the tools

you will use to determine their effect on the servers you manage. A performance tool shouldn't affect monitored data by more than 1 percent to 5 percent.

Understanding the Numbers

Your next task is to gather information and act on that information. The four major components of performance management and tuning that affect UNIX systems are processors, disk and file systems, memory, and network. These areas dramatically affect UNIX's effectiveness in managing crucial business applications. Some components have a greater effect than others (e.g., disk and memory greatly affect database servers). You, as a systems manager, must fully understand how your applications behave and the performance your applications require from the Linux and UNIX servers on which they run.

As I discuss my example J2EE server, you need to consider your own goals. This chapter focuses on low-level values such as processor utilization and memory consumption. When evaluating an application server you also need to consider effective performance (e.g., the time before a user gets a response). End-to-end response time is often called total response time.

Processors

Processors are an important component of servers. However, processors aren't always the most important element to consider when designing and deploying your servers. Memory and disk storage often have the largest significance in performance tuning. Because processors are still important, you need to be familiar with the processor line that your preferred platform uses (e.g., SPARC for Sun, POWER for IBM). You need to understand your processors' strengths and weaknesses, as well as how effectively your processors work in multiprocessor systems (i.e., an SMP server).

Note: UNIX has a long history with RISC processors. A RISC processor fundamentally differs from CISC. RISC processors read fixed-sized instruction words from memory. Ensuring that instructions are a fixed size reduces the complexity necessary for a processor during the fetch and decode stages and can increase the efficiency of the instruction cache on the processor. This benefit can significantly improve how fast a RISC processor processes instructions. RISC processors tend to prefer simple instructions. Thus, RISC processors often support numerous instructions, many of which must combine to perform complex tasks.

Whereas RISC instructions are a fixed size, CISC instructions can be short or long. Variable-length instructions place a heavier burden on the processor during the fetch and decode process than do fixed-size instructions. CISC processors can combine what would be multiple RISC instructions for a common but complex task into one CISC instruction. A CISC processor can do more work than a RISC processor for each instruction the processor fetches, although the CISC processor might be slower at fetching and decoding instructions.

Modern Intel processors have strayed from the CISC approach toward RISC-based processor cores. This change eases design requirements

and often creates a faster processor. However, to ensure compatibility with past processors (and thus the millions of applications that run on those processors), new Intel processors appear to be CISC processors—they support the same instruction set but still operate with an internal RISC core.

Monitoring Tools

Linux and UNIX systems have several tools to monitor processor performance. You need to focus on the server's workload—that is, how much work the processor is performing and whether it's keeping up. A simple measurement is the uptime command, which displays three important values: a 1-, 5-, and 15-minute load average. The load average is the number of processes waiting to run on a processor (i.e., in the run queue).

Following is an example of the uptime command run on a Linux system:

```
# uptime
10:22pm up 116 days, 17:57, 2 users, load average: 0.48, 0.38, 0.43
```

You can use the vmstat command to view the actual run-queue size:

```
# vmstat 5 2
procs      memory      swap    io      system      cpu
r  b  w swpd  free  buff  cache si so bi bo  in cs  us sy id
0  0  0 62800 182884 137052 429856 3 3 1 3 3 0 0 4 3
1  0  0 62800 181376 137052 429884 0 0 8 316 193 161 2 2 96
```

This example shows a run-queue of 1 (see the second line—always ignore vmstat and iostat's first line of output). This value is typically greater than 0. If the value is always 0, your processor is probably faster than necessary for your target application. This situation might be acceptable, depending on your requirements for handling peak loads. Conversely, a system with a high load average might not be overutilizing its processors. Instead, you might have a memory shortage and the processor might be spending a lot of time paging memory to and from disk. (This situation is usually evident when vmstat's output shows a high sy value.) A good rule of thumb is that the run-queue shouldn't be greater than two to four times your number of processors.

Vmstat also displays several other excellent performance indicators. Vmstat's output varies across UNIX systems, but useful values to monitor include r, b, w, cs, us, sy, and id. Table 1 explains these values.

Table 5-3. Important Vmstat Values

Value	Description
r	Processes ready to run but waiting for time on the processor.
b	Processes blocked by I/O.
w	Processes ready to run but swapped out; values larger than 0 indicate a memory shortage.
cs	The number of context switches performed by the OS.
us	Percentage of time spent running application processes.
sy	Percentage of time spent running the kernel.
id	Idle time.

Randomly viewing `vmstat`'s output or checking the output only when a problem occurs isn't a good performance management strategy. Instead, you need to consistently record `vmstat` information. You can use System V's `sar` tool, which Linux systems' `systat` package includes, to record this information, or you can use `cron` to run `vmstat` on a regular basis (15- to 30-minute intervals work well for long-term trending). The following script will record the information you need:

```
#!/usr/bin/sh
PATH=/bin:/usr/bin
LOG=/var/log/performance/vmstat.txt
vmstat 5 3 | tail -1 > $LOG
```

Notice that the script discards the first two lines of output. `lstat` and `vmstat`'s first line of output shows averages since the system started; you can disregard this insignificant information. We disregard one additional line for good measure, and record the last line.

The `sar` tool is useful for long-term performance trending and troubleshooting. You can use `sar` to reduce `vmstat`'s information:

```
# sar
Linux 2.4.20-18.9 (mail.example.com) 04/18/2004

12:00:00 a.m. CPU %user %nice %system %idle
12:10:01 a.m. all 3.60 0.00 1.44 94.96
12:20:00 a.m. all 0.73 0.00 0.47 98.80
12:30:00 a.m. all 0.82 0.00 0.52 98.66
12:40:00 a.m. all 1.24 0.00 0.52 98.24
```

This example shows `sar` information until 12:40 a.m. and breaks down CPU usage by `%user`, `%nice`, `%system`, and `%idle` (Table 2 explains these values). Because no standard exists for `sar` output, available values can differ between UNIX flavors and even between versions of one UNIX flavor. I used Linux's `systat` package to generate the example output; `systat` provides a rich set of `sar` features.

Table 5-4. *Sar CPU Output*

Value	Description	Tips
%user	Percentage of time spent executing at the user level (e.g., applications).	This value should typically be at the highest level (i.e., your application is receiving most of the CPU resources).
%nice	Percentage of time spent executing process at a nice priority (i.e., process run with the <code>nice</code> command).	This value typically isn't high unless you run a lot of nice processes, which is common if you use <code>cron</code> to run jobs.
%system	Percentage of time spent executing at the kernel level.	This value should be low unless you're running an NFS server—high values indicate excessive disk I/O.
%idle	Percentage of time spent idle.	This value should remain at 20 percent to 30 percent—this level leaves room for spikes and indicates a good return on your CPU cost.

As Table 2 explains, %user should typically be the largest of the four values, which indicates that your applications (rather than the OS) are getting the most access to the CPU. If %system is high you should check your disk I/O; a high %system value indicates high disk I/O or memory paging. The %idle rule of thumb is that for the best return on your investment, you don't want a server that idles more often than not. This rule especially applies to application servers (e.g., a J2EE server). You want to maximize CPU use so that %idle is between 20 percent and 30 percent. This level leaves room for spikes and ensures that your processor investment is justified.

Sar gives you important long-term trending data, but you sometimes need information for a specific time period. Top is a useful tool for spot-checking your systems' performance. Tools such as sar are important, but top provides information about several aspects of performance in real time. Top isn't superior to sar—they serve different roles. Top lets you view in real time the resources available and in use on a server, as well as which processes are using your CPU. The following example shows information about processor utilization and processes currently running on the system:

```
10:32 p.m. up 116 days, 18:07, 2 users, load average: 0.89, 0.61, 0.48
291 processes: 288 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 0.5% user, 1.3% system, 0.0% nice, 98.0% idle
Mem: 998908K av, 865896K used, 133012K free, 0K shrd, 137132K buff
Swap: 1052248K av, 62800K used, 989448K free          435440K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
14966	qmailq	17	0	1784	1784	1112	R	0.5	0.1	0:00	qmail-scanner-q
14617	root	12	0	1240	1240	868	R	0.3	0.1	0:00	top
14830	qmaild	15	0	296	296	248	R	0.0	0.0	0:00	qmail-smtpd

In this case I started top and entered i to show only active (i.e., running) processes. The example system was a lightly loaded mail server. Top shows the server's load averages, how many processes were running and sleeping, memory information, and active processes.

Multiprocessor Systems

SMP-based application servers are commonly used because they offer increased processing ability within a single server. Because application servers (e.g., J2EE servers) often perform a considerable amount of computation, their processor use is high. SMP-based systems help alleviate processor-based performance problems but don't provide a total solution. SMP obeys the property of diminishing return because of the communication necessary between processors (assuming that the application is designed to effectively use SMP systems). As you increase the number of processors you get less value from each additional CPU. Depending on the application, the communication and synchronization necessary between processors might override the advantage of an additional processor. In such a case, adding a processor might decrease system performance.

Disk and File Systems

Storage is typically a UNIX server's downfall. Although network bandwidth and latency can cause problems, local storage devices receive the most use in a server environment. Thus, storage must be a major component of your performance tuning and management strategy.

Although disk performance has improved slowly, especially compared with components such as processors and memory, disk capacity has grown rapidly. This trend negatively affects disk performance. The following analogy helps explain this effect.

Consider the speed at which a server can read and write from a disk drive as a window into what that disk stores. The faster you can read and write to the disk, the larger the window—that is, the more you can see at a time. Historically, the size of this window has grown slowly, while the landscape behind the window has grown quickly. Although more information lies behind the window, your ability to see the information hasn't kept pace. Your goal is to increase the size of the window as much as possible. You might need to combine techniques and technologies, including using disk caches or striped disks, to achieve this goal.

Note: UNIX file systems are fast but often slow down when a directory contains numerous files. This decrease in speed can dramatically affect file operations such as a directory listing or search. You can use a hashing-style directory layout to solve this problem. Applications such as the Postfix SMTP server use this technique. For example, you can lay out your application's data storage so that the first few directories are based on the file name, as in the following:

```
/data/0/3/03433
```

```
/data/3/9/39343
```

```
/data/7/4/74934
```

This technique can improve performance if you have a large number of files (e.g., in the case of a Web proxy that maintains a disk-based cache). In addition, consider using file systems that perform well for your target application (e.g., XFS handles large files well and therefore might work best for a computer graphics rendering farm).

SCSI and IDE

Using the correct technology ensures that your performance window is as large as possible. Most non-Intel-based UNIX systems use SCSI as the interface between the server and disk storage. Intel-based systems often let you use SCSI or IDE. Although IDE-based disks have decreased the SCSI performance gap, SCSI still outperforms IDE-based storage, especially for workloads characteristic of UNIX servers.

You can improve performance even if you use IDE (e.g., on a server with minimal disk activity that therefore doesn't require SCSI-level performance). Methods to improve performance vary, but you need to ensure that the IDE disk is operating in the highest performance mode. Some OSs, such as Linux, often operate drives in a low-performance fail-safe mode by default. The assumption is that the administrator knows the drive better than the OS does, so the administrator will set the correct mode. In Linux you can use the `hdparm` tool to reset the mode. For example, to change from 16-bit I/O mode to 32-bit I/O mode you use the `-c 1` option:

```
# hdparm c 1 /dev/hda
```

To display the drive setting, enter:

```
# hdparm /dev/had
```

```
/dev/hda:
multcount      = 16 (on)
I/O support    = 1 (32-bit)
unmaskirq      = 1 (on)
using_dma      = 1 (on)
keepsettings   = 0 (off)
nowerr         = 0 (off)
readonly       = 0 (off)
readahead      = 8 (on)
geometry       = 14593/255/63, sectors = 234441648, start = 0
busstate       = 1 (on)
```

The method you use isn't as important as understanding that the OS often makes the safest but slowest choice when configuring devices. You must carefully evaluate each device on your server to properly configure the devices, particularly for IDE drives.

You typically don't need to set SCSI disks to a high performance mode. The host bus adaptor (HBA), SCSI cable, and physical disk drive determine a SCSI disk's mode (e.g., FastSCSI).

Use IDE if you have low disk I/O needs and minimal random access. SCSI outperforms IDE in the case of multiple applications reading from and writing to disks, especially when this disk use causes random access patterns. SCSI also has high throughput that IDE is only beginning to match.

Note: Fibre Channel (formerly called Fiber Channel but renamed to avoid the perception as fiber-only technology) is a high-speed, high-bandwidth serial protocol that can span several miles. Fibre Channel has a long UNIX history; the technology has often been used to tie UNIX servers to remote storage devices. Although Fibre Channel's popularity is waning because of its expense, the technology can greatly reduce SCSI and IDE's distance limitations. Fibre Channel lets you span several miles, rather than SCSI and IDE's mere meters.

Performance Considerations

Disks can slow system performance. A simple solution is to use a faster disk. In addition, you can use RAID 0 (i.e., disk striping), which can dramatically affect read and write performance. Unfortunately, RAID 0 is risky because each additional disk in a RAID 0 set further reduces the mean time between failures (MTBF). A good use of RAID 0 is for database index files—using RAID 0 on these files can greatly affect performance, and you can usually replace the files if they're lost.

In many cases, properly using memory in place of physical storage is also a good idea. An example is using memory-mapped files. When you memory-map a file (in C, you use the `mmap()` function), the kernel reads the file from disk and places the file into memory. When the application that opened a file (e.g., Tomcat) reads or writes to the file, the kernel can return the file contents from memory rather than disk, which results in a performance gain. If the file is opened several times, the performance improvement compounds.

Another technique for using memory to replace slow storage is the `tmpfs` file system, which is available on OSs such as Solaris. `Tmpfs` is most commonly used for the `/tmp` directory, which read- and write-heavy applications (e.g., mail servers) often use for temporary files. In `tmpfs`, the `/tmp` directory exists only in memory, rather than being a

physical partition on the disk. Thus, reads and writes to these files are as fast as the server's memory.

Monitoring Tools

Linux and UNIX include standard tools that provide long-term trending information for disk performance. Because of disks' importance to UNIX servers, you need to consistently gather and review this information, especially as you add users or expand your applications' scope.

When you bring a new disk online, run performance tests to obtain a baseline of the disk's capabilities. For example, you can use Linux's `hdparm` tool with the `-Tt` option to achieve this goal for IDE disks:

```
# hdparm Tt /dev/hda
```

In a heterogeneous UNIX network you need to use a tool that works across all systems so that you can easily compare and contrast performance across systems and not worry about OS issues. A good disk benchmarking tool is IOzone (<http://www.iozone.org>), which provides in-depth information about disk performance. Whereas `hdparm` supplies two values, IOzone tests your disk against a wide range of read and write sizes to best determine whether a disk is optimized for your application. For example, many databases work best when the storage system can quickly read and write large blocks of data, whereas NetNews servers work best with disks and file systems that are tuned for small files.

For disk utilization monitoring, the `iostat` utility works well, particularly when you collect values over consistent and long time periods. The following code shows the use of `iostat` on a mail server with an IDE disk:

```
# iostat 5 2
Linux 2.4.20-20.7 (mail.example.com) 04/21/2004

avg-cpu:      %user   %nice    %sys    %idle
              4.54     3.48     3.59    88.39

Device:      tps    Blk_read/s  Blk_wrtn/s  Blk_read  Blk_wrtn
dev3-0       37.89   368.69      306.04      3720956812 3088711490

avg-cpu:      %user   %nice    %sys    %idle
              3.80     0.00     1.60    94.60

Device:      tps    Blk_read/s  Blk_wrtn/s  Blk_read  Blk_wrtn
dev3-0       4.20     0.00       188.80       0         944
```

Ignore the first set of output (i.e., the first instance of `avg-cpu` and `Device`). My example shows that the IDE disk device (i.e., `dev3-0`) has very low usage. The transactions per second (tps) value is the number of I/O requests issued to the device. Because UNIX systems, such as Linux, try to combine multiple requests into one transaction, the tps value might not indicate the actual number of requests the drive is processing. However, the tps value is a good indicator of drive utilization—the higher the tps value, the more requests the drive is receiving. The tps value is often over 100 on a server with many small files that requires numerous reads and writes. The `Blk_read/s` and `Blk_wrtn/s` values show how many blocks are read and written to per second; `Blk_read` and `Blk_wrtn` provide the total numbers for the time period. These values vary depending on how you use your disks.

Real-World Performance Tuning

In Linux, the `/proc/sys/fs` interface provides several variables that affect system performance. Displayed values include total allocated file handles, currently used file handles, and maximum file handles that can be allocated. Linux system defaults tend to be low:

```
# sysctl fs.file-nr
fs.file-nr = 7343 2043 8192
```

The Linux kernel dynamically allocates requested file handles but doesn't free handles when they're no longer needed. Instead, Linux recycles file handles. You can view the total allocated file handles to see the maximum number of files used during peak times. Some servers, such as database servers, don't need numerous open files. Other servers, such as mail servers, can have thousands of open files. For a mail server, 8192 maximum file handles (as in the example above) might not be enough. You can update `fs.file-max` to adjust the maximum number of file handles that Linux will allocate:

```
# sysctl -w fs.file-max="32768"
fs.file-max = 32768
# sysctl fs.file-nr
fs.file-nr = 7343 2043 32768
```

In this example I increased the maximum number of file handles that can be allocated from 8192 to 32,768. I would then monitor the total allocated file handles to see if the new value was approached—if so, I would further increase the value. Tomcat and other J2EE servers have many open files, so this optimization greatly affects the total number of requests such servers can service.

Determining `sysctl` variables and available ranges often requires you to read kernel documentation. Although tuning `sysctl` values can be difficult, doing so is often crucial and can result in large performance and reliability gains.

Memory

Insufficient memory on a UNIX server can kill performance. UNIX relies extensively on its virtual memory system to manage core, or primary, memory (i.e., RAM) and secondary memory, which is usually a paging file or disk partition. Modern servers have a lot of memory—usually several hundred megabytes to several gigabytes. Specialized servers, such as database servers, often have much more memory.

Slow memory, like a slow disk, can negatively affect performance. For example, computational clusters often used in genomics or other calculation-heavy processing industries frequently store large arrays of values in memory. As the algorithms for these applications require new data, the information is fetched from memory. Processors have been considerably faster than memory for more than 20 years, and this trend is continuing. You should purchase the fastest memory you can afford.

You also need to consider your memory speed. UNIX relies heavily on a virtual memory system that pages data from memory to disk whenever necessary. Paging typically occurs during a memory shortage or when memory claimed by a process hasn't been accessed for a certain amount of time (which varies depending on the UNIX flavor). Most Linux and UNIX systems have low levels of paging, which indicates that the kernel is maintaining an adequate level of RAM available for new memory requests. Unacceptable paging includes paging because of a memory shortage. In this type of paging, memory

that a process currently needs is given to another process (e.g., when a new program starts). This process causes excessive disk activity and can significantly slow your UNIX systems' performance.

One solution for memory shortage is to redesign your applications to use less memory. You can tune applications for time or space. In tuning for time your main consideration is how long an algorithm takes to execute. Tuning for time typically consumes more memory because your main consideration is how long a process requires to run. Tuning for space causes your algorithm to take longer but consumes less memory. Depending on how your application works, the best approach might be to tune for space in general and tune for time only on an application's most used portions.

Another solution is to add memory to your server. This option is the cheapest because RAM prices are so low. Doubling the amount of memory on a server can have dramatic results and often costs as little as 10 percent to 20 percent of the original server cost. Most UNIX OSs are effective at using additional memory and can therefore reduce or eliminate paging to disk. Because disk paging drains server performance, eliminating paging is a good idea.

You can use `vmstat` or `sar` to monitor memory usage. Although `sar` provides more details, `vmstat` is sufficient if your main concern is memory consumption.

You might occasionally have low levels of sporadic paging. This situation is normal in UNIX because the OS tends to page out memory that hasn't been used for a long time. Consistent paging to and from the disk, however, is cause for concern. Paging considerably slows processes, especially when memory is low and paging occurs often. As I mentioned previously, the best solution is to increase the amount of memory available to the server.

Note: J2EE servers and Java applications in general can consume a lot of memory. Java doesn't explicitly allocate memory, so memory is sometimes in use longer than necessary. As the number of requests increases, memory consumption also increases, placing more demands on the memory and virtual memory system. You need to ensure that your J2EE servers have adequate memory.

Real-World Performance Tuning

Linux and UNIX virtual memory systems are key OS components. In Linux, two variables under `/proc/sys/vm` let you adjust how the disk buffers and the virtual memory works with your disks and file systems. To learn more about these variables and the values you can specify for them, read your Linux OS's documentation in `/usr/src/linux-version/Documentation/sysctl/`, where *version* is your Linux kernel's version. In addition, commercial UNIX vendors' system documentation and Web sites typically provide extensive information about kernel tuning parameters (e.g., Sun Microsystems provides performance tuning information at <http://www.sun.com/bigadmin/features/articles/bestpractices.html>).

`Sysctl's` `vm.bdflush` variable lets you adjust how the kernel flushes dirty buffers to disk. The kernel uses disk buffers to cache data stored on disks, which are slow compared with RAM. When a buffer becomes sufficiently dirty (i.e., the buffer contents no longer match the disk), the Linux kernel daemon `bdflush` flushes the data in the memory buffers to disk.

`Vm.bdflush` has several parameters:

```
# sysctl vm.bdflush
vm.bdflush = 30 500 0 0 500 3000 60 20 0
```

Some of the parameters are dummy values. You need to monitor the first, second, and seventh parameters—`nract`, `ndirty`, and `nract_sync`, respectively. `Nract` specifies the maximum percentage of a buffer that `bdfush` allows before queuing the buffer to be written to disk. `Ndirty` specifies the maximum number of buffers `bdfush` flushes simultaneously. Finally, `nract_sync` is similar to `nract`—but after the percentage that `nract_sync` specifies is reached, a write is forced rather than queued. (Again, refer to `/usr/src/linux-version/Documentation/sysctl/` for technical details.)

Adjusting `vm.bdfush` is complicated because you need to extensively test the effect on your server and target applications. If the server has an intelligent controller and disk, decreasing the total number of flushes (which causes each flush to take longer) might increase performance. However, with a slow disk the system might spend more time waiting for the flush to finish.

The best way to test your changes is to benchmark your target application before and after the changes. Although component-specific testing (e.g., using `IOzone` to test disk storage performance) is excellent for determining the underlying hardware and system's maximum speed, you should always use the application that will run on the server for performance testing. This method is the only way to determine whether your changes will positively or negatively affect your system's performance.

The default values for `nract` and `nract_sync` are 30 percent and 60 percent. When you increase `nract`, ensure that the new value isn't equal to `nract_sync`. In the following example, `nract` is set to 60 percent and `nract_sync` to 80 percent:

```
# sysctl w vm.bdfush="60 500 0 0 500 3000 80 20 0"
vm.bdfush = 60 500 0 0 500 3000 80 20 0
```

The `ndirty` parameter specifies how much of the disk buffers `bdfush` will write to disk at once. The larger this value, the longer `bdfush` might take to complete its updates to disk.

Network

In UNIX, the term *network* generally means TCP/IP. Although you can use UNIX on non-TCP/IP networks, most UNIX software and system services (e.g., `Telnet`) operate over TCP/IP. I focus here on TCP/IP, but many of the principles I discuss also apply to other network technologies.

Networks are described in relation to how they can be mapped to the Open System Interconnection (OSI) model, which consists of the following layers:

- Physical layer—The physical medium that carries bits of information between two systems.
- Data link layer—How data sends over the physical medium; this layer is specific to the hardware technology used (e.g., Ethernet).
- Network layer—How data transports between two endpoints in a network (e.g., IP).
- Transport layer—How the network layer is managed in more detail (e.g., TCP, UDP).
- Session layer, presentation layer, and application layer—These combined layers form the layer at which applications run.

Although the OSI model is useful for discussing networking in an educational setting, this model isn't useful for actually implementing networking technologies. One of OSI's problems is its complexity. Networking technologies such as TCP/IP collapse many OSI layers (e.g., session, presentation, and application). Although this strategy makes network protocols easier to develop, the OSI model still provides a more logical approach to separating functionality between protocols.

Physical Network Performance

Your company's physical organization, rather than your preference, often determines the network technology you use. Companies with only one office can often rely on Fast Ethernet or Gigabit Ethernet-compliant cabling (i.e., Category 5 and Fiber Optic cabling). Offices spread across a campus typically use different types of media. You might use Cat 5 within an office and fiber between buildings. This approach makes sense because it uses inexpensive cabling when possible and relies on expensive physical technologies (e.g., fiber) only when necessary.

Many administrators make the mistake of not establishing a performance management strategy that encompasses the physical layer. Rather than monitor traffic performance (which is better done at a higher layer), you need to check for cable cuts and errors. This monitoring falls under fault management (i.e., when something goes wrong) and performance management (i.e., when performance falls short of needs or expectations). To perform this type of monitoring, you can use tools specifically targeted to detect physical layer problems (e.g., from Fluke—<http://www.fluke.com>).

Many network switches don't properly recognize an Ethernet device's port speed. New servers are often configured to work at Fast Ethernet full-duplex speed. This configuration, which is the fastest possible for a Fast Ethernet interface, is preferable. However, switches often don't detect this setting and instead use half-duplex or only 10Mbps performance. The UNIX interface then switches to the slower speed. Although you have the infrastructure for improved performance, your switch opts to not use it.

You might want to disable autonegotiation and instead lock the switch port to Fast Ethernet full-duplex if your UNIX server's interface can support that mode. If you make this change, you need to document it so that someone else doesn't later clean the switch's configuration and reset it to autonegotiation.

Note: Various Linux and UNIX systems manage their interfaces differently. For example, in Linux you can use `mii-tool` to manage network interface characteristics (e.g., half- or full-duplex mode). Other UNIX systems use different tools and configuration files to accomplish this task.

TCP/IP Performance

You also need to evaluate how well you're using TCP/IP to ensure your applications' optimal performance over the network. Rather than discuss best practices, I focus on rules of thumb that provide good performance for target applications.

You need to understand TCP/IP's various relationships and how these relationships affect performance. DNS greatly affects your applications' performance and availability. If you have a slow or unresponsive DNS server, your applications might time out when they try to connect to remote servers. Your performance management framework must address possible infrastructure problems.

Monitor Protocols in Use

Network performance management should monitor which protocols are most prevalent on your network. For example, you might be surprised to learn that you have more NetBEUI traffic than you originally anticipated (especially if your network still has many Windows 98 and 95 systems). You might then try to eradicate the extra traffic from your network (which is a good goal in the case of NetBEUI), or you might optimize your network for the generated traffic.

Note: The NetBEUI example is intentionally contrived. Network managers often fail to monitor the protocol use on their networks, only to later find suboptimal performance. You should always know what traffic you're supporting, how that traffic affects the network technology you're using, and how you need to organize your network topology to accommodate the traffic. For NetBEUI, a good tactic is to minimize your subnetworks' size and use routers between workgroups whenever possible. This strategy eliminates the burden of frequent NetBEUI broadcasts overrunning the network.

An easy-to-use open-source tool for monitoring which network protocols are in use is ntop (<http://www.ntop.org>). This tool can give you a detailed protocol history and show which computers are generating the most traffic. More comprehensive commercial packages, such as Network Instruments' Observer (<http://www.networkinstruments.com>), also exist for monitoring network protocol use.

Focus on Important Applications

Related to network protocols use, you also need to know which applications are running on your UNIX servers and how those applications interact with the network. Application use greatly affects performance, particularly during slow response times.

Categorizing applications into support and production roles is useful. Support applications, such as NFS, can place a large burden on the network and other applications' performance. Support applications tend to be more infrastructure-related than production applications are. Production applications are typically those applications that end users use or that support your company's daily business activities. For example, an Oracle database or a J2EE server is a production application.

A common performance management mistake is to place too much emphasis on one application type, such as Web traffic, and too little on another, such as NFS traffic supporting the Web servers. Systems administrators often tune their systems specifically for support applications. For example, with NFS, ensuring that large packets are sent is beneficial because this strategy reduces protocol overhead and therefore increases efficiency. You need to consider how this approach affects the other services you support. Your goal is to tune according to your most important applications.

Bandwidth vs. Latency

A struggle will always exist between how much data you can push and the latency involved. The applications you need to support often affect this discrepancy. For example, if your primary concern is end-user response time (i.e., how fast users perceive an application to run), then latency will drive your performance tuning.

Segment Your Networks

Before network switches became common, organizations often used routers to separate networks based not only on physical requirements but also on performance needs. That is, servers that needed high levels of bandwidth to communicate with one another were placed on one network, and other systems, such as those that end users used, were kept on a separate network so that the high-traffic network didn't directly affect them. This approach is still valid.

One way to separate networks in this manner is to place routers between them; another method is to use Virtual LANs (VLANs) in switches. Figure 1 shows an example of using VLANs to separate switches to isolate traffic. One switch is a high-performance Gigabit Ethernet switch. These switches are expensive, and workstations underutilize them. However, dedicating servers to a Gigabit Ethernet switch (particularly servers such as NFS servers that transfer large amounts of data to other servers) yields a large performance increase. Systems that don't need this level of performance are placed on a separate switch. This approach is cost effective because you can spend less money on systems that don't need a high level of service.

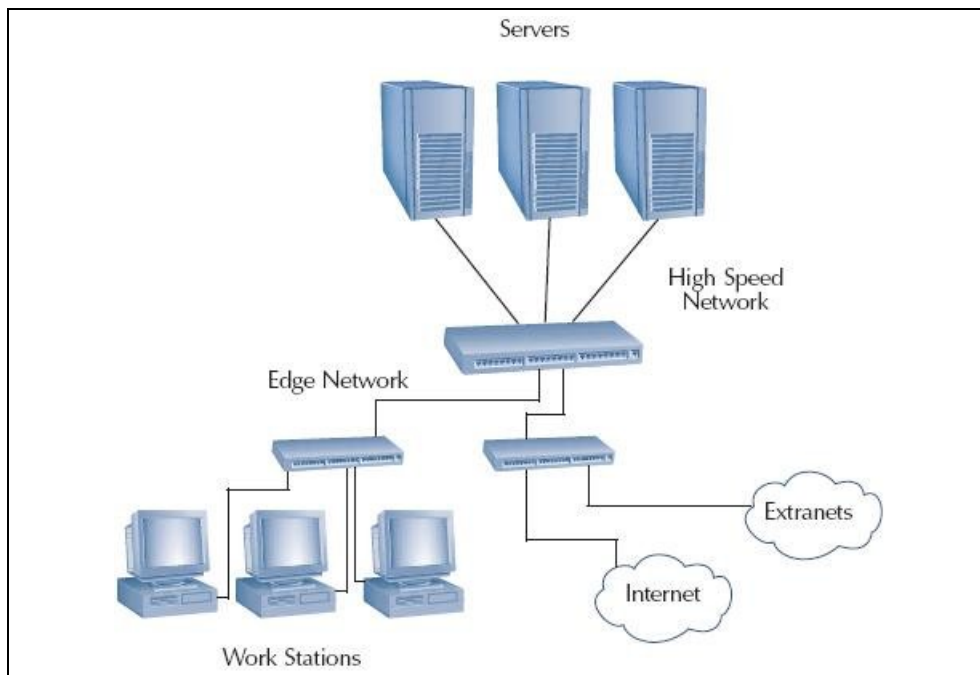


Figure 5-27. Separate Networks by Traffic

Load-Balance When Possible

One of the best ways to ensure that you have the resources you need during normal and peak loads is to load-balance across multiple servers. Although scaling a system up (i.e., increasing one server's power) has benefits, scaling out lets you handle large loads and gives you physical redundancy. Even if a server fails, your service remains available.

Because J2EE services use HTTP for communication (or HTTP over Secure Sockets Layer—SSL), J2EE servers behave similarly to Web servers. Thus, you can use existing Web-clustering products (e.g., F5) to easily load-balance these servers (e.g., Tomcat)

across several machines. You can also use Round Robin DNS (RRDNS) to load-share, but this approach doesn't work as well as true load-sharing, particularly when you consider DNS caching.

F5 Networks (<http://www.f5.com>) and Coyote Point Systems (<http://www.coyotepoint.com>) offer a range of load-balancing devices. These devices are designed specifically for load-balancing and ensuring high availability. They automatically stop the flow of traffic to servers that go offline, thus dramatically improving your network services' (e.g., J2EE, LDAP) performance over load-sharing techniques such as RRDNS. You can also use open-source technologies such as the Linux Virtual Server (<http://www.linuxvirtualserver.org>) to offer load-balancing and high availability.

Real-World Performance Tuning

On a server that has frequent but short-term connections, a concern is the number of half-open connections that can be maintained. A half-open connection occurs when a TCP connection is initiated but not yet completed. These connections are placed into a queue for later processing. In Linux, you can view netstat's output to see these connections:

```
# netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp      0      1 10.0.0.1:25    24.34.2.3:3434 SYN_RECV
```

This example shows one half-open connection, in which the client is 24.34.2.3 and the server is 10.0.0.1. Because 10.0.0.1 is a private range addresses (i.e., not routable over the Internet), a router must be performing Network Address Translation (NAT) between 10.0.0.1 and 24.34.2.3. In addition, because the remote client is accessing port 25, 10.0.0.1 is probably an SMTP server (which it in fact is).

SMTP servers, like Web servers, incur many connections with only a small amount of data transferred. Most emails range in size from 4KB to 8KB, which is a small percentage of the maximum TCP packet size. You need to be concerned about half-open connections if your network traffic consists of small packets with frequent connections.

Suppose you decided to compromise between the memory that the buffers consume while maintaining half-open connections and the speed at which you want to be able to accept new connections, especially when the server is busy. You might decide you want to be able to support as many as 1024 simultaneous half-open connections. You can use the sysctl tool to increase the number of possible half-open connections:

```
# sysctl -w net.ipv4.tcp_max_syn_backlog="1024"
net.ipv4.tcp_max_syn_backlog = 1024
```

This change deals with new incoming connections, but what if you need to connect to another server? For example, a J2EE server might need to connect to a database server or other servers. UNIX OSs, including Linux, typically define a range of ports reserved for outgoing connections. As a connection occurs, one of the ports in this range is consumed. If the connection isn't closed, one less port is available. Linux's default range is 1024 to 4999, but you can increase this range (according to `/usr/src/linux-version/Documentation/sysctl`):

```
# sysctl -w net.ipv4.ip_local_port_range="32768 61000"
net.ipv4.ip_local_port_range = 32768 61000
```

This change lets the application server make 28,232 (61,000 minus 32,768) rather than 3975 (4999 minus 1024) outgoing connections.

TCP's efficiency is closely related to the TCP congestion window size. The TCP congestion window is the amount of data that can be sent between two systems before an acknowledge (ACK) is necessary. The larger the window the less overhead needed to ensure that packets reach their destinations. Little or no network data loss occurs in a LAN environment, so you can increase the congestion window as much as possible to boost performance (i.e., to approximately 64KB).

Modern TCP/IP stacks try to optimize the congestion window. The window starts small and grows as the two servers increasingly trust the link between them. Setting the congestion window to a large size initially is sometimes preferable, especially for short-term connections that might not last long enough for the window to open to an efficient size. The congestion window's maximum size is the size of the send buffer that the kernel maintains. The send and receive buffers store data, which is then placed into TCP/IP packets and sent to the remote host. You need to increase the send and receive buffers to increase the congestion window.

In Linux, you can use `sysctl` to modify the values `net.core.wmem_max` and `net.ipv4.tcp_wmem`. The `net.core.wmem_max` value specifies the send queue's maximum buffer size for any protocol, including IP version 4 (IPv4). The `net.ipv4.tcp_wmem` variable includes three parameters: the minimum buffer size regardless of how much stress is on the memory system, the default buffer size, and the maximum buffer size. The default size that `net.ipv4.tcp_wmem` specifies overrides `net.core.wmem_default`, so you can ignore `net.core.wmem_default`. In addition, `net.core.wmem_max` overrides the maximum buffer size that `net.ipv4.tcp_wmem` specifies. When changing `net.ipv4.tcp_wmem`, ensure that `net.core.wmem_max`'s maximum buffer size is as large as or larger than `net.ipv4.tcp_wmem`'s maximum buffer size. (These values typically range from 8KB to 128KB.)

To change Linux's minimum guaranteed buffer from 4KB to 16KB and maximum buffer size to 128KB, change both values. First, examine the original values, as the following shows:

```
# sysctl net.ipv4.tcp_wmem
net.ipv4.tcp_wmem = 4096 16384 131072
```

You can use the bandwidth-delay product to determine your optimal window. This calculation helps you determine a general range in which to experiment with congestion window sizes:

$$\text{windows size} = \text{bandwidth (bytes/sec)} * \text{round-trip time (seconds)}$$

Suppose you determine that your optimal congestion window size is 48KB. You then need to adjust `net.ipv4.tcp_wmem`'s parameters to show 48KB (i.e., 49,152 bytes) as the send buffer's default size:

```
# sysctl -w net.ipv4.tcp_wmem="4096 49152 131072"
net.ipv4.tcp_wmem = 4096 49152 131072
```

Historically, the congestion window was limited to 64KB. Request for Comments (RFC) 1323 removed this limitation with the introduction of window scaling, which allows even larger values. Although TCP window scaling is enabled by default on the 2.4 kernel, if you specify a value of 64KB or larger you need to manually ensure that windows scaling is enabled:

```
# sysctl -w net.ipv4.tcp_window_scaling="1"
net.ipv4.tcp_window_scaling = 1
# sysctl -w net.core.wmem_max="262144"
net.core.wmem_max = 262144
# sysctl -w net.ipv4.tcp_wmem="4096 131072 262144"
net.ipv4.tcp_wmem = 4096 131072 262144
```

In the example I increased the default buffer size to 128KB and the maximum buffer size to 256KB. The new default and maximum buffer sizes are larger than net.core.wmem_max's original values, so I also adjusted the net.core.wmem_max value because this value overrides net.ipv4.tcp_wmem's maximum.

Other values to consider include net.core.rmem_default, net.core.rmem_max, and net.ipv4.tcp_rmem. These values control the receive buffer's size and can greatly affect client systems and file servers.

Performance tuning is complicated and requires a firm grasp of Linux and UNIX fundamentals. Linux and UNIX system defaults are typically sufficient for workstations but are rarely adequate for a highly loaded server.

Conclusion

Understanding your UNIX server's performance isn't enough. As a systems manager you must help define, measure, and enforce your network's performance requirements. You must first ensure that you can review historical performance information for all your UNIX servers. The best method for accomplishing this task is to save your servers' performance data to a central database. The tools vmstat and iostat let you easily gather and store this information. A more advanced method is to use an agent-based performance management tool that collects a wide range of information. A commercial tool's greatest benefit is often its ability to warn you in advance of possible problems. Commercial tools also include detailed and readable performance analysis reports that you can use for ad-hoc or long-term reporting to management.

6

User Management

Linux and UNIX have a strong set of services to support networks, including well-tuned and reliable TCP/IP stacks and network services (e.g., Secure Shell—SSH—for remote administration). Indeed, the Internet—the largest network of all—is built on a backbone of UNIX-based servers.

Managing user accounts across networks in a UNIX environment is difficult because of the dual need for security and reliability—if user accounts become compromised or unavailable, companies lose money. Even non-networked UNIX servers require account management (e.g., SCO OpenServer is well established in the medical industry as a single, console-based server and often isn't networked). But UNIX's real account management difficulty occurs when the OS is networked.

This chapter discusses UNIX's account management needs, including the strategies that enterprises use to manage those accounts. Remember that most enterprise environments have multiple UNIX flavors, as well as non-UNIX OSs such as Windows. Thus you need to clearly define *user account* (or *digital identity*) and identify the information necessary for an account to be functional across these systems.

Account Management Elements

User management primarily involves UNIX system users and the accounts those people use to gain system and network access. Such account information includes users' account names, passwords, and user files. In UNIX this information falls into four key categories:

- Account information
- Group membership
- Account passwords
- Home directories

Although group membership technically falls under account information, I use the preceding four categories because they are easy to map into specific files that UNIX maintains. For example, the `/etc/passwd` file typically contains account information, the

/etc/groups file contains group membership, the /etc/passwd file (or the shadow password file) contains account passwords, and the local client machine or a file server (e.g., an NFS server) contains home directories.

Managing account information and group membership tends to be easy, whereas managing account passwords and home directories is more difficult. In this chapter I discuss technologies and practices for managing account information, group membership, account passwords, and home directories. No rigid rules exist for administering these four components of UNIX user management. Instead, you must select the technology that best fits your organization. You might need to combine solutions to suit your network's needs.

Note: Linux and UNIX systems keep user passwords in shadow password files. Securing /etc/passwd and /etc/group against brute force attacks is difficult because these files must be world readable. To overcome this weakness, shadow password files store passwords in files that only the root user can read. Because the /etc/group file can contain passwords, most UNIX systems support this file with a shadow file. For example, Linux uses the shadow file /etc/gshadow to support the /etc/group file. You can use gpasswd to administer /etc/gshadow.

Account Information

When we discuss users, we mean not only the people but also their accounts. Each person is unique but can have multiple accounts. Although restricting each person to one account might be preferable, legacy and political concerns often prevent this practice.

Account information typically includes a user's real name and username and might include other information, such as when the user can log on. Most users can't edit their own account information and typically must request changes from an administrator. Therefore, account information rarely changes and is easily manageable from a central resource. Using Network Information Service (NIS), Lightweight Directory Access Protocol (LDAP), or replication from a master to slaves (i.e., servers and workstations) is effective for managing account information.

Note: NIS is excellent for distributing user information but is insecure, as I discuss later in this chapter. NIS+, the next generation of NIS, is more secure but differs significantly from NIS. Because you can't easily upgrade from NIS to NIS+, you might instead consider another technology (e.g., LDAP).

Group Membership

Group membership is as important as account information. In UNIX, group membership often dictates a user's rights to system files. Because UNIX is so file centric, group membership also determines general user account access. UNIX groups don't always play a role inside UNIX applications (e.g., a human resources—HR—application), but some systems, particularly those that work with LDAP, can use groups to manage rights both outside and inside applications.

Note: A common design flaw in many UNIX systems is that administrators place new users in a common group (e.g., the Users group). This practice lets new users easily share information because a

user must set group-read and group-write permissions only on the files he or she owns. However, placing all users into a one group is dangerous. If a user's umask is set to an inappropriate value (e.g., 007), all users on the system (i.e., all users in the Users group) can read and write the files that user writes.

Linux offers a safer alternative. Most Linux distributions create a new group for each new user. Users must then specifically let others access their files.

Account Passwords

Account passwords are one of the most problematic aspects of user management. One problem is that users prefer to change their own passwords rather than requesting an administrator to do so. Ideally, you need to let users update their passwords on any machine they log on to. Unfortunately, implementing this ability can be difficult. Instead, you might employ one server or application (e.g., a Web application) for users to update their passwords. Another problem with passwords and user management is passwords' sensitivity. Exposing passwords to users is risky. Some systems, such as NIS, are particularly poor at password management.

Note: Many UNIX networks are moving to Kerberos for user authentication. Kerberos is relatively secure and has industrywide acceptance.

Home Directories

UNIX stores files in user home directories. If users need to log on to multiple servers, you need to make the users' home directories easily accessible. Most enterprises use NFS servers to store user home directories and an automounter to mount the directories to the servers the users log on to.

Note: Although NFS is UDP-based and stateless so that you can lose an NFS server and still survive, clustering NFS remains a difficult task. The problem with NFS clustering is that most UNIX systems that mount an NFS file system can't easily handle a failed server. Indeed, unmounting an NFS file system on some systems (e.g., FreeBSD) can be almost impossible if the NFS server is down. Some of the large commercial NFS servers offer clustering services. In critical environments you should ensure that your NFS servers are stable and reliable and that you can cluster NFS servers and hide the clustering from the NFS clients.

Centralized Management Using NIS

One of the most prevalent and pivotal solutions for UNIX account information management is NIS. NIS is based on the distribution of *maps*, which are files that NIS pushes to clients. NIS lets you have master, slave, and client servers. As Figure 1 shows, the NIS master is the central map location, the slaves provide redundancy and can be queried by clients, and the clients are simply workstations and servers using NIS for user

authentication and information. One of NIS's most important maps is the passwd map, which lets you centrally manage user accounts and passwords.

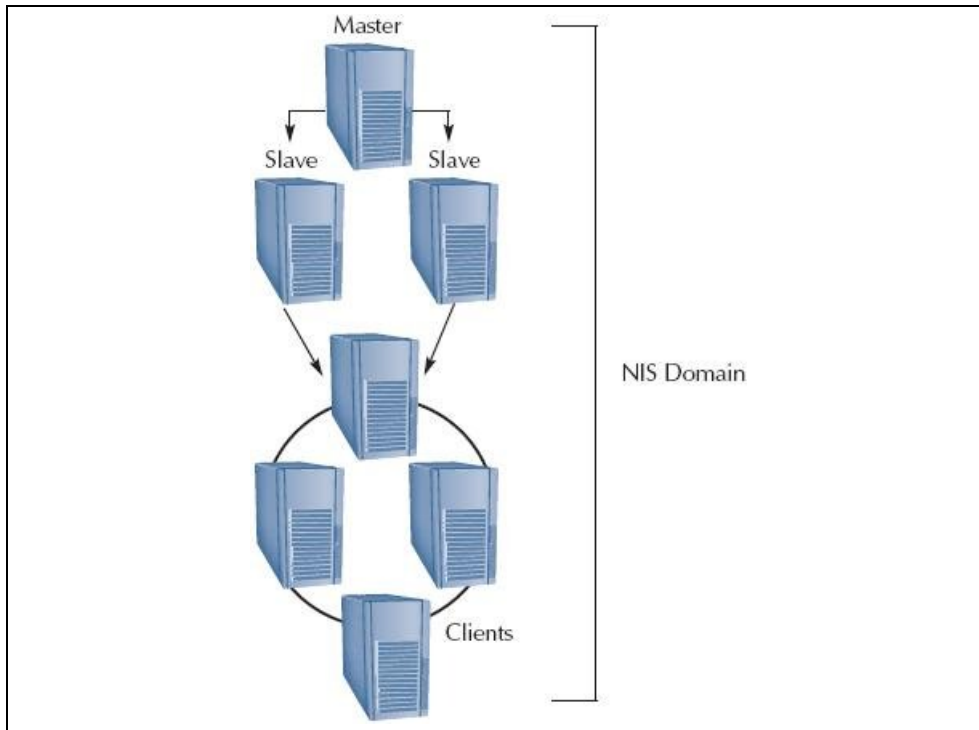


Figure 6-28. NIS Masters, Slaves, and Clients

Using NIS provides several benefits, including the ability to centralize your user accounts to one system. Centralizing accounts eliminates the problem of keeping multiple accounts updated across all your UNIX servers (e.g., by manually updating the password file). NIS is also Linux and UNIX's best-supported account management system. But NIS's ultimate downfall is its insecurity.

NIS's insecurity stems from two major design flaws: lack of encryption and lack of authorization. NIS traffic isn't encrypted between the server and clients; thus, any NIS maps that cross your network can be sniffed. In addition, any NIS domain member has the authorization to download any of the maps in the domain, including the passwd map. Because passwords in NIS password maps must use a relatively weak form of encryption (i.e., crypt), regular users can easily steal user accounts and passwords in an NIS network. You should limit NIS use for user passwords to only the most secure and trusted environments.

Despite NIS's problems, many UNIX administrators still use this solution and even deploy it in new environments. Thus you need to consider the following NIS guidelines and best practices.

NIS Domain Names

Although an NIS domain name doesn't need to be the same as a network's DNS domain name, having the same name is often preferable. Because attackers can easily determine

NIS domain names, using a hidden or secret name offers little advantage. Mapping your NIS domain name to your DNS domain name keeps management simple. However, this practice prevents you from subdividing your NIS domains unless you also do so with DNS.

Redundancy

You need to maintain NIS slaves. When no NIS servers are available on the network, UNIX systems that use NIS start to fail—particularly for services such as email and interactive logons. Each NIS domain needs a master and at least one slave. Slaves also let you better distribute the load of serving NIS maps. In general, you need one NIS server for every 40 to 60 NIS clients. This range varies depending on the services you run and your real-world load. Having more clients per NIS server can dramatically reduce the response time to map query requests, which significantly affects network performance.

Note: Having too many servers that provide redundancy to an important application such as NIS can create as large a burden as having too few servers. If you support numerous NIS slaves, many of your resources must manage the infrastructure that drives your account management rather than offer users more tangible services such as additional drive space or improved spam filtering. Quantify your network's needs and include room for peak load, but don't go overboard when providing redundant NIS slaves.

NIS and Windows

This book assumes that you manage several OSs. You most likely support not only enterprise UNIX networks but also Windows networks. You need to consider how to reduce the complexity and cost of managing two very different types of environments.

One potentially cost-saving strategy is to use NIS to integrate UNIX and Windows. You can create your own custom tools to integrate NIS and Windows, or you can use Microsoft's Services for UNIX. SFU includes software that lets Active Directory (AD) DCs also serve as NIS masters. Configuring this software and joining the servers' NIS domain lets you integrate your UNIX systems into an existing AD network. This technique is transparent to your UNIX users and lets you easily manage both environments.

Note: You can also use Samba (<http://www.samba.org>) to integrate UNIX and Windows. Samba provides file- and print-sharing services and includes software that can seamlessly integrate Linux and UNIX systems into an existing Windows NT domain or AD network.

Securing NIS

One of NIS's biggest drawbacks is its lack of security. However, administrators continue to use and deploy NIS. Thus, you need to consider how to eliminate or minimize NIS's security shortcomings.

Restrict Network Access

Because true authentication doesn't occur between servers in an NIS domain, you must ensure that only authorized hosts have access to the network containing the NIS domain. First, place routers and firewalls between the NIS domain and external servers to guard the NIS domain's perimeter. To protect the network, your router shouldn't allow source-based routing and IP spoofing and should block remote procedure call (RPC) going into and out of the network. (Source-based routing should always be disabled.) Also restrict or eliminate any incoming and outgoing connections that aren't explicitly allowed (e.g., SSH for UNIX administration).

In addition to restricting access to NIS networks, restrict access to NIS servers and clients. Because NIS lets anyone in an NIS domain view maps, you need to limit the number of people who can log on to NIS servers.

You can further strengthen NIS's security by ensuring that NIS servers respond only to authorized NIS clients. You can't make this limitation directly in NIS, but you can use Wietse Venema's `tcp_wrappers` (which I discuss in Chapter 2) to do so. Specifically, use `tcp_wrappers` to protect access to the portmapper.

Restrict Access to NIS Masters and Slaves

NIS masters contain the crucial information necessary to push maps to each of the hosts in an NIS domain. You need to restrict access to the NIS masters and slaves, much as you restrict access to important servers (e.g., DCs in an AD network). Restrict access to systems administrators only—regular users don't need direct access to the NIS masters and slaves.

Note: In protecting NIS masters, you need to consider the NIS data's confidentiality, integrity, and availability. Security is based on these three factors. In general, a compromised NIS client exposes only the confidentiality of NIS information, whereas a compromised NIS master or slave can result in the loss of confidentiality, integrity, and availability.

Patch Systems

Properly patch your NIS masters, slaves, and clients. Much of the NIS code in use today has existed for a long time, and most of the code's obvious vulnerabilities have been removed. However, additional vulnerabilities are possible. You need to constantly maintain the patch levels on your NIS servers and clients.

Shadow Passwords and NIS

One of the NIS's main problems is its lack of true support for shadow passwords. NIS networks typically support only crypt-based passwords, which are vulnerable to brute force attacks. This problem is particularly important because anyone can view the NIS `passwd` map. A brute force attack can create serious security risks.

No standardized method exists for using shadow passwords with NIS. Your UNIX version might or might not allow shadow passwords. If you have a heterogeneous network, shadow passwords might be nearly impossible. Consult your UNIX version's documentation.

Centralized Management Using LDAP

NIS is a good choice for its original purpose (i.e., map distribution). However, NIS's security shortcomings, especially related to over-the-wire encryption, make it too risky to use in large enterprise networks. You need to constantly monitor NIS and ensure proper functioning; yet NIS still exposes passwords to brute force attacks. A better solution is to use an alternative that centralizes information as NIS does but provides more security.

When considering centralized user management, pay special attention to two core needs: accessing account information (e.g., usernames) and authentication. You can combine account management and authentication (as in LDAP), or you can keep management and authentication separate and use several solutions' best features to address the problem's components. If you separate management and authentication, you might use NIS or replication to distribute maps and user information, then use Kerberos for authentication. This method lets you use both solutions' best features. I focus on using LDAP as a solution for combining account management and authentication, thus centralizing your information and reducing the overall work necessary to use and support the service.

Note: If you use LDAP for sensitive services on your network (e.g., user authentication), you need to use LDAP over SSL (LDAPS). You can use LDAPS or LDAP with StartTLS. In LDAPS, LDAP is wrapped in a Secure Sockets Layer (SSL) connection. In StartTLS, LDAP begins over an unencrypted channel, then switches to an SSL mode shortly afterward to protect sensitive information. Both modes are equally secure, although LDAPS tends to be better supported by third-party applications and network appliances.

The Directory Service

Solid account management, and more generally, identity management systems, rely on a common directory service to maintain and offer identities to applications and OSs. To implement identity management, you must decide on a directory service. Current choices include LDAP, AD, and Novell's eDirectory; of these, only LDAP is completely open. Thus LDAP is a good choice for a long-term platform on which to base your enterprise's future. AD and eDirectory are solid, lasting solutions; however, open standards are preferable for long-term decisions.

One of the most important attributes for a directory service that you'll use in conjunction with an identity management system is the ability to exactly create and manage attributes that belong to entries. As your organization grows, you'll encounter many applications that require specific attributes, and you might want to customize the information stores within identities. To achieve these tasks effectively in a directory, you must be able to specify attributes that contain the information you need.

The directory service you use will incur a heavy load. Be sure that the service you select has solid search performance (i.e., many servers can perform search queries against your directory without the service slowing down), scalability, and the ability to replicate data between LDAP master servers and slave servers for performance and high availability (which ties closely with scalability). Also ensure that you can integrate the service with your target applications.

Location of Directories

As you determine the directory service to use and consider your directory structure, you must decide where to locate the directory service inside your network. Your location choices include enterprise and perimeter. An enterprise directory is housed and managed in the enterprise network's core and most often acts as the central identity management directory service that powers the company's identity and access management. A perimeter directory service is used on the perimeter (e.g., demilitarized zone—DMZ, extranet). Perimeter directories are at greater risk for being compromised; thus, you need to restrict the information that these directories maintain.

An enterprise directory typically provides a central source for user accounts, as well as secure storage for authentication credentials (e.g., passwords). In addition, enterprise directories often contain information about network resources (e.g., printers), listings of people (e.g., a company address book), and group relationships. Finally, enterprise directories are often closely linked with enterprise networks' security services and technologies.

Perimeter or extranet directories fill a different role. These directories provide application support for partners and customers, and they often support public-facing services such as mail or Web site authorization and access. Whereas an enterprise directory provides one consistent view, a perimeter directory must be able to represent several partners' and customer groups' needs and information.

Because external users (e.g., partners, customers) often outnumber internal users, perimeter directories often have higher loads than internal, enterprise directories. This increased load also means that perimeter directories have additional licensing requirements.

Directory Management

UNIX directories must be able to supply access management services (i.e., authentication and authorization). Before deciding on a directory infrastructure, you must consider user authentication requirements, including how those requirements will affect your solution. For example, what authentication requirements will you place on users, and what technologies will you need to do so? Although directories provide centralized user management, using a directory to achieve single sign-on (SSO) is difficult. You also need to consider whether your applications support using the directory for authentication. Determine your enterprise authentication needs, including how your policies dictate that users access your systems and networks, as well as your need for complete control of authentication and authorization.

Some authentication services, when combined with directory services, offer enterprise users an SSO experience. The best example is Kerberos used for authentication. Applications in turn use existing Kerberos tickets for authenticated users, along with access control information in directories, to provide a full-featured SSO experience.

However, using Kerberos for external clients (e.g., customers, partners) can be difficult. In addition, perimeter networks often have different requirements. For example, you don't need as much control over user access to services because you're offering only the services you want users to have. User access is often limited to Web-based applications. You therefore need to be realistic about the kind of access control you can place on users. Application-specific methods such as Web-based forms or digital certificates often

authenticate users. You might also use an authentication service such as Microsoft Passport, which can offer users a form of SSO.

Laying Out the Directory

LDAP directories are hierarchically structured. When building your directory, ensure the required level of separation between containers (e.g., Accounts and People) but don't make the directory overly complex. A common directory-structuring problem is how to handle user accounts. (This problem also involves identity management, which I discuss later in the chapter.) In many organizations each user has one account to use throughout the organization. However, this situation isn't always feasible. For example, consider legacy systems that you must use a new directory to support. Such a legacy system might require a specific account name length, type of account name, or other restriction. But you might be able to use just one account for all other applications. In such a situation you probably need to allocate multiple accounts for each user.

Typical LDAP directories have two containers: People and Accounts, as Figure 2 shows. The People container has entries representing each physical person in the organization. These entries are usually of the type `inetOrgPerson` and might contain information specific to the organization. The Accounts container includes the user accounts that each person in the organization uses to access his or her applications.

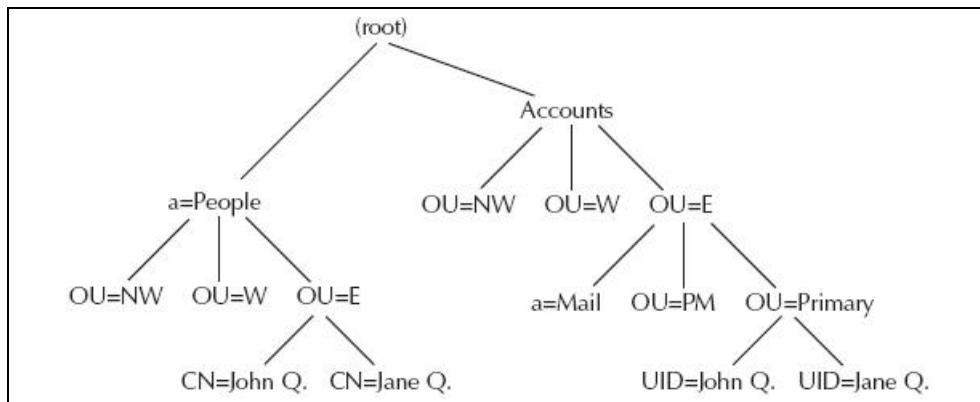


Figure 6-29. People and Accounts Directories

The People container can include only people entries, or you can divide People into additional containers. These subcontainers might represent a geographical or political separation of people. As in Figure 2, the People container might include the objects Northwest, West, and East, showing the company's divisions. Subdividing your containers lets you delegate management of each division to the appropriate people in the division.

The Accounts container is often more complicated than the People container, depending on your organization's and applications' needs. Figure 2 shows an organization with the Accounts container split into geographical divisions to support delegated authority, and the East division is further separated into three main areas: Mail, PM, and Primary. The Mail container is for mail accounts. The example organization in Figure 2, which is a medium-size healthcare provider, still supports POP3 for remote users without VPN and therefore prefers to keep its mail accounts and passwords separate from the general accounts. This technique ensures that theft of the POP3 password won't compromise

other applications. The PM container is for the company's practice management software. This software has been modified to support LDAP, but the internal code still supports only very small usernames. Because only a subset of users need the PM accounts, the company decided to simply provide a specialized account for each user rather than incur the expense of an extensive modification. Finally, the Primary container houses the users' primary accounts. Users use their primary accounts for logons, VPN access, and access to other applications. Alternatively, you could place the primary accounts directory under OU=E and ensure that applications using LDAP didn't descend further into the directory (e.g., into OU=PM) when searching for user accounts.

The structure that Figure 2 depicts is more complex than if each person had only one user account. However, this more complicated configuration is better suited to the example organization's real-world needs.

This example shows several best practices in effect. First, the organization doesn't use high-vulnerability accounts (e.g., the POP3 account) for other applications. The company applied the principle of least privilege; the POP3 account has access only to the user's mail. Second, the technology is adapted to the business rather than vice versa. Many companies make extensive changes to suit one application or method. In this case the organization used LDAP's adaptability to its advantage.

Note: The layout of the directory in Figure 2 is complex. In general, you need to simplify and flatten the layout of a directory as much as possible. However, many applications require you to create specialized containers in LDAP to support those applications.

LDAP, Linux, and UNIX

You can seamlessly integrate Linux and most UNIX flavors into an LDAP infrastructure. For example, Linux supports `pam_ldap`, which is a Pluggable Authentication Module (PAM) that lets Linux perform authentication and account information lookups in the background. As users log on and perform file-system operations such as directory listing searches using `ls`, Linux performs lookups against an LDAP directory. You can also configure other server components, such as the print system, to use LDAP.

Note: You typically use vendor-supplied tools to configure Linux to use LDAP (e.g., for user authentication). For example, you can use `authconfig`, the authentication configuration application, to configure LDAP support for Red Hat systems. This ability demonstrates Linux and UNIX's support for LDAP.

Distributed User Management

Both NIS and LDAP let you centralize user account information and password management. Another option is distributed user management. Distributed user management lets you maintain a central database of user information, then push or pull that information to various systems and applications. Although this approach might seem identical to centralized management because information is kept in a central location, this method is different because remote systems and applications use their own internal formats to authenticate and authorize users. In a centralized model, remote systems and applications request authentication and authorization services from NIS or LDAP.

The distributed user management model's primary benefit is that you don't need to change your servers' and applications' configurations. This advantage is particularly useful if you can't update the systems you support (e.g., a large inventory-control application that supports only a local user database).

Distributed user management relies on a central location for management and information, as Figure 3 shows. You can then push or pull information to the necessary applications. In the push method, you initiate updates from the master server. In the pull method, the client periodically polls the master server for changes. An advantage of the push method is that applications receive changes immediately. Unfortunately, the push method requires the central server to maintain each remote server's state. If a push fails, the central server must retry the update later. The pull method prevents this problem because the remote server always knows its own state. However, the pull method doesn't let you obtain updates immediately. You might use both methods depending on your application mix.

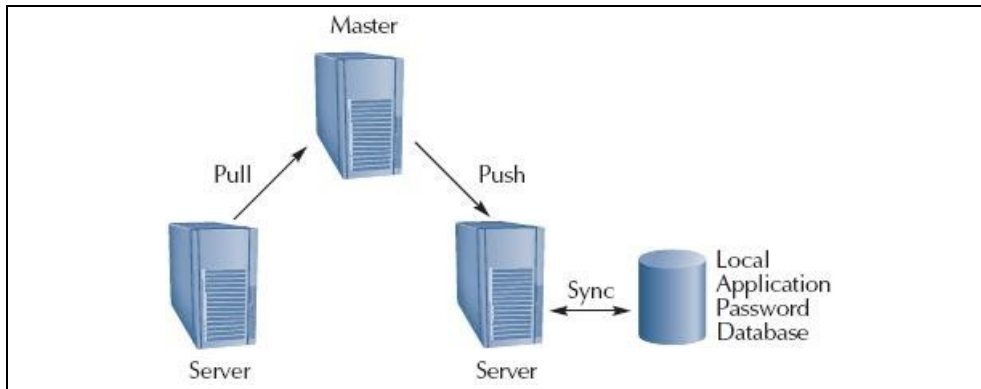


Figure 6-30. Distributed User Management

Another benefit of distributed user management is the ability to integrate several platforms, including UNIX, Windows, and other platforms and applications. As long as the local application can read the central data, the local machine can run on the distributed user management framework.

Note: Distributed user management is very offline friendly. That is, you can easily take systems running within a distributed user management infrastructure off an enterprise network, and the systems can still serve local users. This benefit is particularly useful in a large, geographically separated network in which maintaining constant online access to a core network (e.g., the network providing a Kerberos authentication service) is impractical.

The User Management Policy

Now that you understand the various methods for managing user accounts, we can outline a user management policy. I focus on a high-level user management policy that meets a UNIX environment's core needs. Although I don't delve into such a policy's details, the example that Figure 4 shows provides an outline of a real-world user management policy.

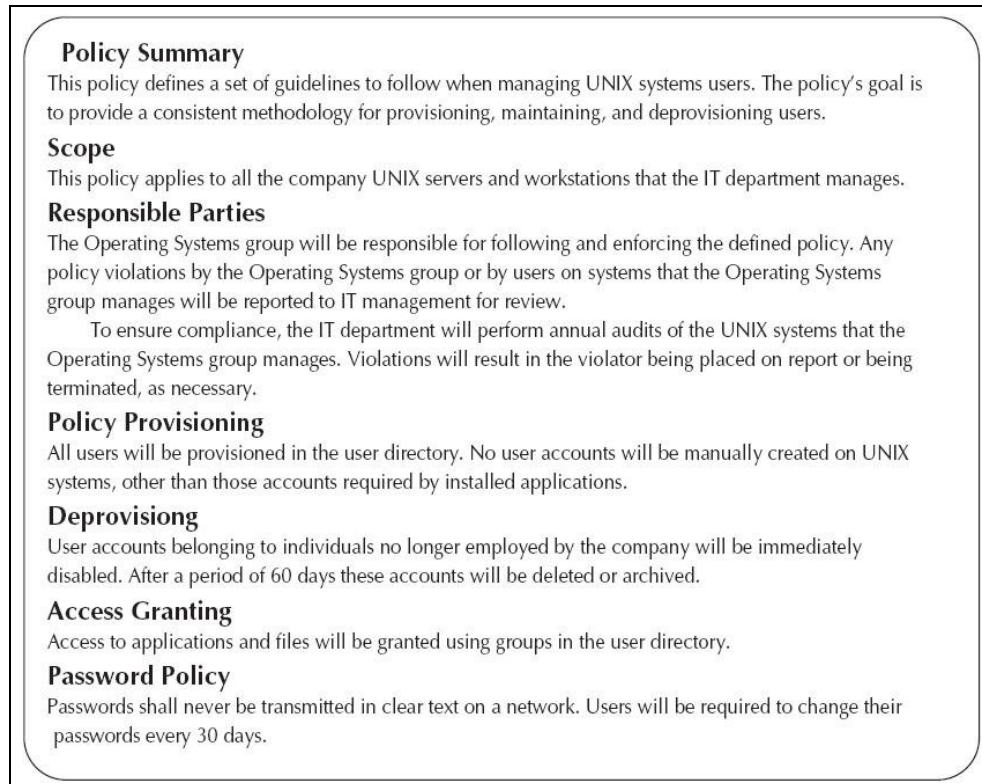


Figure 6-31. Example User Management Policy

More About Identity and Access Management

Identity management is related to the larger topic of user management. All users require digital identities to access network and system resources across UNIX and other systems. Unfortunately, identity management is difficult if only because finding a solution that works across all platforms and applications is difficult. This task is particularly problematic in large enterprise environments, in which acquisitions greatly affect the business.

A consistent identity and access management infrastructure is important. You need to plan for the strategies and technologies necessary to implement the appropriate level of identity and access management in your organization. Solid long-term planning is especially crucial for cost-effectiveness. Your solution shouldn't cost more than it's worth, impede the daily business activities that drive the enterprise, or hinder long-term projects or future acquisitions.

A reliable and consistent identity management policy lets you centralize identity management. An enterprise with ad hoc or no identity management can't easily integrate the systems within its infrastructure.

A consistent identity management solution can dramatically reduce your enterprise's administration and technology requirements. Using fewer technologies lets you consolidate your employees' education and experience. In addition, you can then use one

method to implement identity management across all your applications. LDAP is an excellent example of a system that gives you one reference point for identities and has widespread industry acceptance.

Finally, a well-documented and well-accepted identity management solution lets you more easily integrate new acquisitions and business partner systems into your e-infrastructure. All enterprises will eventually need to integrate their IT and identity and access management with partner systems. Using industry-standard solutions eases this task.

Identity Management

In addition to directory management, you need to consider identity management. All OSs come with tools to manage identities based on how the OS implements identities. For large-scale enterprise-level identity management, you might need a more comprehensive open-source or commercial solution. Such a solution should address at least user and authorization management.

Identity Integration

You often can't merge all your identities into one comprehensive directory. For example, some products require application-specific directories. You might be able to use a comprehensive management product to manage these directories with a metadirectory integration. The identity management product you use must integrate with your existing application directories. In addition, determine whether the software requires its own user account in the directory that it manages (preferably not). Finally, make sure you can use rules or customization to extend the integration software. Over time the number and variety of directories you support will increase (e.g., after a merger); you need to use tools that will grow with your company.

Conclusion

User management is an important part of UNIX management. User management involves provisioning and deprovisioning users, maintaining account information in a centralized directory, securing passwords, and ensuring consistent and reliable access to user files. When designing and implementing your infrastructure, consider the most flexible approach you can use (e.g., LDAP) so your technology easily scales with your business.

7

Fault Management

Fault management includes detecting, reporting, and reacting to undesired events. Such events on UNIX systems can range from disk failure to kernel panic. For any undesired event, the systems administrator's goal is to quickly determine the affected machine and minimize or eliminate downtime. This chapter discusses the UNIX fault management problems that systems administrators and managers face. Figure 1 shows the previously mentioned three steps involved in fault management: detecting, reporting, and reacting.

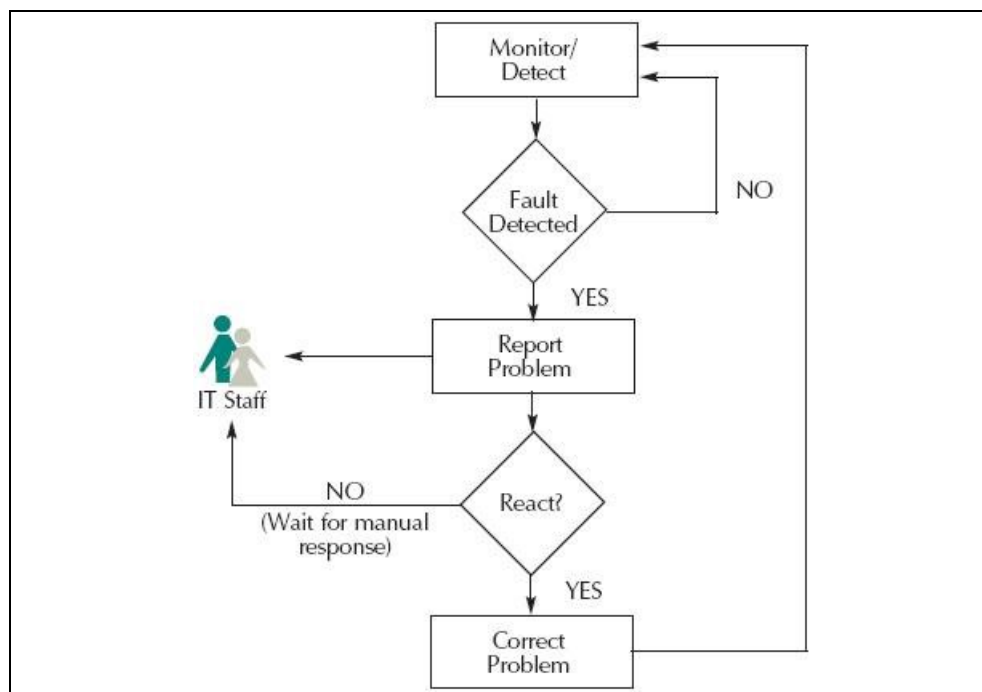


Figure 7-32. Detecting, Reporting, and Reacting to Faults

Fault detection means ensuring that a system or monitoring tool captures undesirable events such as disk failure, poor application performance, or loss of network connectivity. Some events (e.g., disk failure) have instant and obvious effects (e.g., data loss). Other types of events (e.g., poor application performance) are difficult to detect because their effects are difficult to precisely define.

After an event occurs, the monitoring system must report the event to a systems administrator or operator. Common reporting methods include emails, pages, and audible computer beeps. Event reporting can be problematic if the reporting mechanism requires a failed component (e.g., if the disk that contains `/usr/bin` becomes unavailable and the reporting system uses `/usr/bin/sendmail` to send email alerts). Because local events can cause reporting systems to malfunction, external machines need to perform the monitoring in case a monitored machine can't report its own problems.

Note: Some vendors' (e.g., IBM's) high-end systems are self-healing and can automatically report problems. Using a vendor-supplied comprehensive fault management system might be beneficial because vendors know best how their systems work and how those systems fail. However, if your network includes equipment from several vendors you need to weigh the benefit of using vendors' fault management systems against the problem of supporting multiple vendor software and management tools. You might prefer to use a third-party monitoring system that works with any hardware and UNIX flavor (e.g., Tivoli, NetIQ).

Reacting to an event is far more difficult than detecting or reporting an event. One of the main problems is knowing what action to take when an event occurs. For example, in the event of a failed disk, should the system automatically activate a hot spare? What if you wanted the hot spare to be dedicated in case the disk housing the human resources (HR) database failed, but the system used the hot spare to replace a low-priority disk? Automating responses to faults can be helpful or harmful.

Components

Faults that occur at the server level are component faults. These events occur because of an undesired change in one of the system's hardware components (e.g., CPU, disk, I/O device). I focus on the two types of hardware that cause the most problems for UNIX systems managers: disks and network interfaces.

Disks

Most UNIX systems rely heavily on locally attached storage (i.e., disk drives). These devices are less reliable than other types of physical server components (i.e., they have a lower mean time between failures—MTBF). In the following sections I discuss various storage problems, methods for detecting storage faults, and common responses.

Note: Enterprise networks are increasingly using Storage Area Networks (SANs). A SAN is a high-speed subnetwork of shared storage devices; that is, a SAN is a specialized network that servers use to access shared storage. Systems administrators often focus on locally attached storage when they evaluate storage problems. If you use a

SAN, you can consider the SAN connection a combined network device and storage device. Thus, you can mix and match the detection, reporting, and reaction strategies I discuss.

Disk Failure

Disks cause more problems than any other system components. Thus, disk errors cause most of the fault events in enterprise environments. Disk reliability has increased in recent years with the advent of technologies such as error-correcting (or self-correcting) disks. Self-correcting disks detect and prevent problems on the platters where data is stored. However, self-correcting disks can't always determine when faults occur. In addition, foreseeing that a disk will fail is more difficult than detecting that a disk has already failed. Ideal systems management means being proactive rather than reactive.

You typically use vendor-supplied software to monitor disks and detect faults. In addition, many UNIX systems have standardized methods for monitoring components. For example, Linux uses syslog to report problems. Thus, you can use a syslog log-scanning program such as Logcheck or Logwatch to automate disk fault checking.

After you detect and report a disk failure, your reaction will depend on the type of fault. On a mission-critical system, a disk or file-system failure might require you to immediately take the system offline for repair. Such an action can be expensive because of lost productivity. A better solution is to provide redundancy that protects against faults in crucial components.

Several RAID levels (e.g., RAID 1, RAID 5) provide disk redundancy. With RAID 1, a mirror of a disk (or disks) is maintained on a second disk. Both disks are generally in use at once, and each disk is available to assume full control if the other disk fails.

Using RAID increases reliability at a storage level but not at a disk level. If both disks in a RAID array fail, the entire array is compromised. You need to know when a disk fails so that you can report and react to the failure.

Vendor-supplied RAID status tools can help you detect RAID array failures. Some systems, such as Linux and FreeBSD, also include tools that you can use to monitor hardware (e.g., Adaptec, Dell PERC) and software (e.g., Linux software RAID). When the tool detects a failure, the hardware might beep or the monitoring software might send you an email alert that contains a syslog log file entry documenting the failure.

Note

No single method exists for detecting RAID array problems. The best method depends on the product you're using. In some cases (e.g., Linux software RAID) no vendor exists. In such a case you can inspect the `/proc/mdstat` file to determine whether a disk fault exists. The `/proc/mdstat` file contents are as follows:

```
# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hda3[0] hdd3[1]
8385856 blocks [2/2] [UU]
```

A capital F will appear next to the RAID device listing if a failure occurs. The following example shows a failed software RAID array:

```
# cat /proc/mdstat
```

```
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hda3[0] hdd3[1] (F)
8385856 blocks [2/2] [U_]
```

Note that U_ replaces UU; this change denotes that the second disk failed. Detecting the (F) lets you automate failure detection, and reading which U is replaced with _ lets you determine the failed disk.

Some organizations use mirroring or replication across hosts to provide for redundancy of not only disks but also systems. This additional redundancy is beneficial, but you must still monitor and test the mirroring process. Monitoring important processes, especially the mirroring of your crucial data, is a best practice. You need to be able to quickly detect and correct problems that occur in your mirroring and replication processes.

Free Disk Space

In addition to disk failure, another cause of lost services is filled disks. Many UNIX installations don't include disk capacity usage monitoring. A filled disk can cause applications to fail or to continue running but to behave improperly (e.g., corrupting information stored on the disk as the application attempts to perform updates). At a minimum, a system needs to monitor the number of free blocks for each file system on a disk. Monitoring inode usage is also a good idea, because a file system with free disk space still can't create new files if no inodes are available. (Some specialized file systems—e.g., Veritas' VxFS—can create inodes dynamically, so you'll never run out of inodes as long as the file system has free space.)

Note: An inode is a UNIX file system data structure that contains information about files. For example, an inode contains information such as who owns a file, the file's access mode, and the file type (e.g., a device file). When you create a UNIX file system on most UNIX flavors, UNIX creates a finite number of inodes. Regardless of how much free space a file system has, if no inodes are available UNIX can't create new files.

The most common tool for monitoring free disk space and inodes is `df`. You can include `df` in a reporting system (e.g., a system that runs nightly). If the free space is below 15 percent, the system should generate an alert and you should take reactive measures. Some UNIX file systems exhibit poor performance when they become 90 percent or more full. Alerting at 85 percent full gives you time to react before problems occur.

Note: You need to know as early as possible when a system's free capacity is decreasing. You can then remove unnecessary files before all the free space is gone. If you don't take action soon enough, the reporting system might fail because it can no longer function correctly (e.g., if the reporting system relies on email, and `/var/spool` is full).

Most UNIX file systems reserve a percentage of the disk for root user use and hide that capacity from normal users. Thus, `df` might report that a file system is 100 percent full when the system is only 95 percent or 97 percent full. UNIX will refuse to let normal users create new files or increase the size of existing files, although the root user will be able to perform those operations. This situation lets the root user correct a full disk before the system crashes. Don't abuse this margin; you need to take immediate corrective action if `df` reports that the system is full.

Disk and File System Performance

I discussed performance management in detail in Chapter 5. Again, I stress the importance of properly monitoring your disks' and file systems' performance characteristics. Over time, users often shift their system usage, which stresses different system components. These effects are especially apparent on storage systems, as the location and method of data storage on disks changes. You need to create file systems for specific tasks (e.g., a file system with large blocks and few inodes for storing large database tables, a file system with small blocks and many inodes for a NetNews server). Using different types of file systems for different tasks is helpful. For example, you might want to use XFS for a file system hosting large files (e.g., a graphics processing server) or ReiserFS for a file system with many small files (e.g., a Usenet news server). You also need to monitor parameters such as disk utilization and the average size of transfers between the disk and server. This information increases your ability to trend performance usage and to develop solutions based on your needs.

Administrators are often surprised when a key application begins to perform poorly. Maintaining consistent, reliable, and detailed performance information for all your crucial servers will prevent this situation. A strong performance monitoring system lets you trend and report on potential problems before they occur.

Note: Even if you don't use a comprehensive, enterprisewide performance monitoring system, you need to regularly use a tool such as `iostat` or `sar` so that you know how your systems typically behave.

When storage performance problems occur, typical reactions are to split the load across multiple disks or servers, upgrade the existing server, or reduce the overall load. Splitting the load across multiple servers is the most expensive solution. This option requires proper application architecture (i.e., the application must be able to run on multiple machines). In addition, you must purchase additional servers. Reducing the overall load is the most difficult solution. This option requires major application changes or restricting application access. Restricting access often causes economic and political problems.

Note: An excellent way to improve disk performance is to dedicate one disk or RAID device to an application. This solution lets the disk and disk cache service only one mission-critical application. If you use IDE (which is an uncommon solution for mission-critical, enterprise-class applications), ensure that the IDE disks don't have the same IDE channel.

Network Interfaces

Networking is integral to UNIX. Because UNIX is so prevalent in server-client computing, you need to instantly detect and report networking faults to minimize reaction time.

To set up and monitor UNIX network interfaces, you need to fully understand networking. Purchasing a commercial solution that provides a full suite of monitoring and testing functionality is often a good idea. You can also build your own toolset of testing and monitoring systems.

To assess networks and network monitoring, you need to consider the following areas:

- Physical (cables, network cards)

- Data link (Ethernet)
- Network (IP)
- Transport (TCP, UDP)
- Application (SMTP, FTP)

These areas coincide with five of the seven Open System Interconnection (OSI) layers. (For a definition of and more information about OSI, see <http://www.webopedia.com/term/o/osi.html> or http://en.wikipedia.org/wiki/open_systems_interconnect.)

Physical Layer

Network devices, unlike disk devices, have a long history of high reliability. Network devices are more reliable than disk devices because network devices have no moving parts. These devices tend to be solid state electronics, and the connecting media are simple cables. Two common network problems that occur at the physical level are cable disconnection and bad cables.

With cable disconnection, a network cable is inadvertently disconnected from a network card. All UNIX systems provide administrators with information about whether media is connected. Even if the primary cable is unplugged, you can easily monitor the interface's status if you have constant access to a server. However, accessing a system is often impossible if the network cable is disconnected. Thus, you need to have an external monitoring service that detects when a host is down. (I discuss such services later in the chapter.)

Note: Disconnected cables are a surprisingly common occurrence, because cables tend to be mislabeled over time in corporate data centers. You need to ensure that your cables are properly tagged and protected. (For example, use cable ladders to organize cables and to stop the cables from coming loose and prevent people from tripping over the cables.)

With bad cables, the connection might work properly but data transfer is slow. This problem is simple to diagnose on a networking technology such as Ethernet. Monitor the network interface transmit and receive errors to determine whether the network card or cable has potential faults. The following output shows no transmit or receive errors:

```
# ifconfig eth0
eth0 Link encap:Ethernet Hwaddr 00:B0:D0:20:04:1A
  inet addr:192.168.1.9 Bcast:192.168.1.255 Mask:255.255.255.0
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:9735033 errors:0 dropped:0 overruns:0 frame:0
  TX packets:8524657 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:100
  RX bytes:2674779201 (2550.8 Mb) TX bytes:2587092355 (2467.2 Mb)
  Interrupt:11 Base address:0xecc0 Memory:fe102000-fe102038
```

Note: Many vendors provide physical layer testing and monitoring equipment. Fluke (<http://www.fluke.com>) is well known for its testing tools.

Data Link and Network Layer

On most UNIX systems you need to monitor the underlying connectivity between hosts at the network or IP layer. Most commercial and open-source network monitoring solutions provide this service. Such solutions typically ping the remote host to ensure that the host can respond at the IP layer. To perform this type of network test, run the ping command.

```
$ ping c 1 host
```

This simple test confirms that the server is on the IP network and can send an Internet Control Message Protocol (ICMP) packet to the receiving hosts. To respond to a ping, a server must have a properly configured network interface and routing table (at least for the local network). If the receiving hosts aren't on the LAN, the routing table must include a gateway for external networks.

Alternatively, a server can ping another host to test itself. This method is common for systems in a cluster. If a system detects that it can't reach other hosts, the system will release the resources it holds (e.g., files on a shared SCSI disk), try to report an error, and perhaps power down.

Reporting an error can be difficult for a server checking its own network connection. If the server doesn't have network access, it can't generate an SNMP trap or email a report to an operator. The only options are to generate a hardware-based report (e.g., the server might beep), use a modem to dial out, or use a serial cable to report the problem. Because servers can't easily report their own network problems, having an external network monitoring service is preferable.

Transport Layer

The transport layer is where most data is packaged and sent. This layer most commonly uses UDP and TCP. One of the most important considerations in testing the transport layer is detecting TCP connection problems.

With TCP, a connection is established and each subsequent packet is sent with a sequence number. If the peer system doesn't respond within a preset amount of time, a timeout occurs and the TCP session closes. This situation usually doesn't occur on a LAN. A physical network defect typically causes TCP sessions to drop (e.g., frequent database connections dropping). Such a situation can occur over a WAN or the Internet because of a slow or troublesome connection. If your TCP sessions are dropping, you need to determine whether the problem is occurring at your end, the remote site's end, or somewhere in between.

Application Layer

The application layer truly controls the network. This layer determines how applications communicate with one another over the network. Common application layer protocols include HTTP for the Web, SMTP for mail, and various database protocols. The other layers can't exist without the application layer. This layer is where business information flows between systems.

The application layer is more complicated to evaluate than the other layers. For example, with SMTP you need to confirm that a connection is possible (i.e., you have a reliable transport layer) and that the application (e.g., Sendmail) is responsive. One of computing's constraints is that only a finite set of requests can be handled at once. Most applications handle this restriction by letting UNIX accept a connection from a client but

not service the request until enough resources (e.g., children) are available to process the request. On an overloaded SMTP server, your network-layer connection will probably succeed but you won't receive the SMTP banner immediately. In such a case, clients will timeout or end users will experience a noticeable service delay.

When you test the application layer, don't test from the server hosting the service—or at least have a redundant external monitor available on the network. Otherwise you might miss network problems (e.g., a poorly configured firewall on the server). Figure 2 shows several methods of application layer testing. Tests 1 and 2 show ineffective self-testing methods.

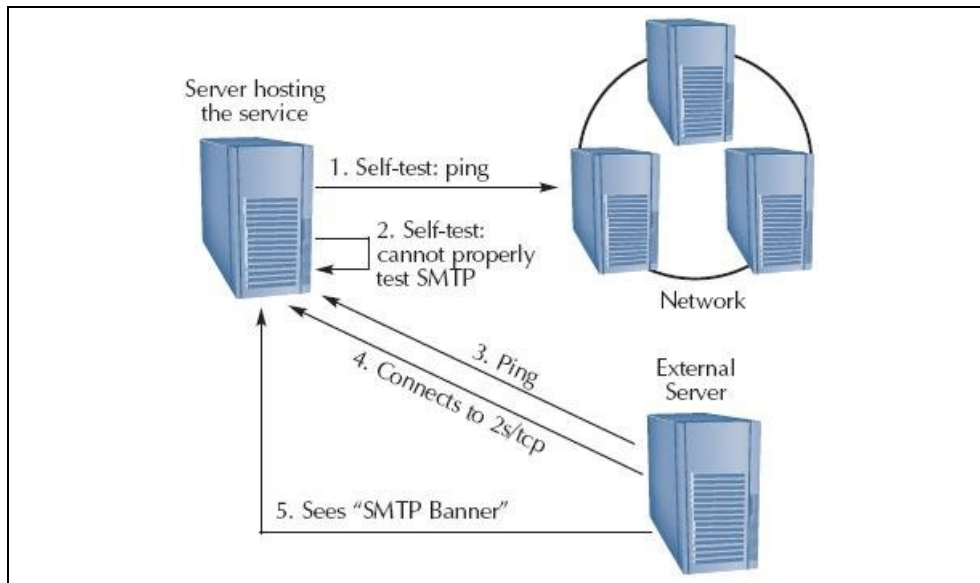


Figure 7-33. SMTP Application Layer Testing

Because of the complexities of detecting errors, a more comprehensive system must be used to monitor for faults. For SMTP we then have the minimal checklist below, which Figure 2 also shows as steps (3), (4), and (5), respectively:

- Is the SMTP host available on the network?
- Does the SMTP host accept a connection on port 25/TCP?
- Does the SMTP host present an SMTP banner?

If you encounter problems with any of these steps, you've detected a fault. Depending on the service in question, you might prefer an automatic or manual response. For example, if the SMTP service fails (e.g., Sendmail crashes) you might want to automate a restart. Alternatively, you might prefer to bring up a redundant service that has multiple SMTP servers with fail-over capabilities.

Testing other applications (e.g., Web servers) is similar: Ensure that the host is available and accepts connection and that the server application is responsive. You can script these actions in a custom monitoring system. Many commercial monitoring packages and open-source software solutions (e.g., Ethan Galstad's Nagios) also have scripted monitoring for common services such as SMTP, POP3, and HTTP.

Another cause of application layer problems is network performance, which I discussed in Chapter 5. Many systems administrators don't properly monitor network performance or don't have an automated monitoring system that alerts them to performance problems. Although performance monitoring data can provide useful information for diagnosing application sluggishness, administrators often mistakenly overlook this data.

When monitoring network performance, you need to consider how latency and bandwidth affect your target applications and users. Latency often has the biggest affect in WAN environments, particularly on applications designed for LAN environments.

Applications

Despite systems administrators' efforts at protecting key UNIX components and services, they frequently neglect important applications. Administrators often ignore applications that are crucial to the enterprise as they focus on disk quotas, memory utilization, and disk faults. However, you need to design, deploy, and manage your servers to provide ultimate application availability.

Monitoring applications involves configuration management, resource usage, performance management (which I discussed in Chapter 5), and fault management. Poorly made configuration changes can greatly affect application performance and availability, whereas poor resource usage can cause an application to fail during high loads. In addition, application bugs can cause faults (i.e., undesired events such as program crashes or database corruption).

Configuration Management

In Chapter 4 I discussed configuration management's importance for ensuring server availability. Configuration management is even more crucial to application availability. Badly written configurations can cause mission-critical applications to fail in obvious or subtle ways.

You use text-based files to configure most UNIX applications. A few UNIX applications have configuration information in a database system such as Oracle or MySQL. When you monitor configurations, you need to monitor not only server configurations but also application configuration files.

Monitoring and managing configuration changes for applications that store configuration information in databases is difficult. A good practice is to perform routine database dumps of the configuration tables, then compare the data in those tables with Last Known Good copies. This method lets you quickly detect unauthorized changes; if necessary, you can use the Last Known Good copies to reload the application configuration.

Resource Usage

Because many UNIX applications are complex, monitoring resource usage can be difficult. Some application resource measurement capabilities (e.g., Application Resource Measurement) provide application developers with a standard API that lets administrators measure performance.

In addition, administrators can use standard UNIX tools to monitor application resource usage. The tools `ps` and `top` tell you how much real and virtual memory applications are using, as well as summarize CPU usage. The tool `du` is useful for determining disk usage.

Fault Management

Application faults involve internal problems—most commonly, bugs in the software code that composes the application program. For example, applications written in C are well known for having problems related to memory pointer and buffer overflow use. In most situations, pointer memory problems result in a segmentation fault (i.e., a `segfault`—UNIX’s method for killing an application with a memory bug) and the UNIX system terminates the application. You can typically use three methods to monitor application faults: monitor application logs, ensure proper application response, and determine whether an application has died.

Note: Bugs in applications that deal with memory can be extremely dangerous to your system’s security. Improper validation of input that’s placed into an application’s memory causes a buffer overflow. Buffer overflows are one of the main causes of application and server failure. The paper “Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade,” by Cowan, Wagle, Pu, Beattie, and Walpole provides an overview of buffer overflow problems and offers solutions.

See http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=821514.

Most applications use `syslog` or a local log file for logging. You can use a tool such as `Logcheck` to monitor these log files and determine whether the application reports any problems (e.g., a corrupt data file). You might also want to automate application-supplied verification tools to operate on a weekly basis. For example, FLEXquarters’ DataFlex database management system (DBMS) provides a tool to scan and verify its database files’ integrity.

To monitor proper application response, you need to define the appropriate response to a given input, then ensure that response is generated. This task is difficult to automate for many applications. For example, if an application is X Window system–based, you might need to script solutions in a language that works with X (e.g., `Tcl/Tk`).

You should always determine whether an application has failed completely (i.e., has died). You can use the `ps` command to create Bourne or Perl scripts that monitor whether an application’s processes are running. Alternatively, monitoring systems such as Quest’s Big Brother (<http://quest.com/bigbrother>) let you automate this process.

Depending on the type of application you’re monitoring, you might want to automate your reaction to a fault. A typical reaction is for an alert to send to an administrator and for the application to restart. (Windows 2000 uses this method to automatically restart failed services.) Some monitoring packages also provide a method for monitoring and automating responses to application faults.

Centralized Monitoring

When you design and build an infrastructure to detect faults, you need to decide whether to deploy customized solutions for each system that you manage or use a centralized, network-based monitoring system. Most businesses initially use the first method. As systems administrators deploy new servers, the administrators script monitoring tools to watch important areas such as resource usage and whether an application is running. This solution works short term but becomes unwieldy as you add servers and applications. Using a centralized fault detection product is preferable. I discuss centralized, network-based monitoring systems in more detail in Chapter 5.

At a minimum, centralized monitoring includes detecting and reporting faults. When a fault occurs, the monitoring system should immediately log the incident and report the problem to IT staff. The system might also include restart capabilities or provide a scripting interface so that the administrator can initiate a series of actions on the server with the failed service. Advanced systems might also let you automate responses to faults. Common network monitoring tools include software such as NetIQ's AppManager Suite, IBM's Tivoli Monitoring, Hewlett-Packard's (HP's) OpenView, Quest's Big Brother, and Nagios.

Note: You also need to determine what to do if your centralized monitoring service fails. If this situation occurs, you might miss other event conditions (e.g., your SQL server's disk fills up and you don't receive an alert). Consider who is watching the watcher: Provide redundancy in your monitoring framework, or at least have another monitoring server that ensures the centralized monitoring service is active and available.

Conclusion

UNIX fault management is complex. You need to consider the various requirements of supported components (e.g., hardware, OSs, services, applications). Your first priority must be to gather baseline data from your crucial servers and services, then build a solid framework to monitor and report faults such as failed disks and crashed applications. After you establish a reporting system, you can automate your responses to faults.

8

Task Automation

Managing a complex network can be tedious and repetitive. Automation lets you reduce the amount of labor necessary to manage your network so that you have more time to perform other tasks, such as increase security and minimize other problems (e.g., administrator mistakes). In addition, automation reduces complexity and creates self-documenting processes that manage systems.

Automation is important for small and large networks. Companies with small networks and small IT staffs have limited man-hours for performing day-to-day tasks and long-term planning. Saving time is important for these companies so that they can better manage their networks. Large companies benefit from automation because they can focus on important upgrades and changes.

In this chapter I discuss automation best practices, how to apply these best practices, and how automation helps manage networks. I begin with a review of automation's benefits and a discussion about when not to automate. Then I outline the automation language's needs and I discuss how to properly automate tasks.

Reasons to Automate

Automation has many advantages. In the following sections I discuss the cost of repetitive tasks and unnecessary complexity, and I explain the benefit of creating self-documenting solutions.

Eliminating Repetitive Tasks

One of the best reasons to automate tasks is to reduce wasted time. Systems administrators often repeat tasks (sometimes incorrectly). Eliminating repetitive tasks lets administrators perform more important tasks, such as improving server efficiency or researching innovative systems management methods (e.g., by reading papers about systems management from USENIX and SAGE).

Reducing Complexity

UNIX networks' complexity makes manually performing tasks difficult. Individual UNIX tasks are simple to complete, but systems management involves executing several consecutive tasks. Most administrators are careful to perform tasks properly the first few times. But after performing the same tasks for a long period of time, administrators often rely on their memory rather than documented procedures. Human nature is to seek the easiest and fastest method for completing a job, which can lead to human error and system errors in the case of UNIX administration. (In addition, typographical errors are common even when you follow instructions carefully.)

Automation is useful for completing many complex administration tasks. For example, consider the task of creating a file system on a newly installed disk in BSD. You need to perform disk testing on the disk, create slices on the new disk, create a file system on each slice, and add each file-system definition to `/etc/fstab`. Although this task doesn't occur frequently, it's sufficiently difficult to warrant automation when possible. In this case you might want to reduce the task to one script that requires only the disk device name and a set of sizes for each slice:

```
# newdisk.sh /dev/ad2 256mb 1024mb 32mb
```

The `newdisk.sh` script takes a disk device and a set of file-system sizes and automates the four steps I mentioned: checks the disk for bad blocks, creates slices, creates a file system on each slice, and adds each file-system definition to `/etc/fstab`. Using one automated script is easier than performing each task separately.

To further reduce complexity, you can delegate senior-level tasks to junior administrators. For example, you can code the `newdisk.sh` script to ensure that `/dev/ad2` isn't a disk in use and prevent possible mistakes. This script lets novice BSD administrators easily create new file systems on a freshly installed disk.

Note: Using tools such as the `newdisk.sh` script lets you push multiple tasks to the Help desk. Help desk administrators then spend less time on trivial tasks and more time responding to problems.

Documenting Tasks

Automating tasks can serve as a form of documentation. If you want a task performed a certain way (e.g., adding users, creating file systems, auditing log files), you can create a script to perform the work and self-document the process. New administrators can then review your scripts to see how various tasks are performed. Even if alternative methods exist for performing a task, your scripts demonstrate how your organization wants the task accomplished. Adding comments to your scripts further strengthens their documentation ability.

You can also use scripts to ensure that administrative tasks are properly logged. Using scripts to automate tasks helps enforce logging while administrative actions are performed. (For more information about logging from scripts, see the section "Log from Scripts.")

When Not to Automate

You don't need to automate all your tasks, because some tasks occur so infrequently or are so trivial that they don't warrant automation. For example, a system reboot might be too trivial to automate. Most systems let you use a *shutdown* or *reboot* command to reboot a system; the reboot is typically logged in *wtmp*. Alternatively, you might want to log the reason for a reboot; in this case, you can automate the task and incorporate logging.

Use One Scripting Language

Large networks often have large and diverse staffs that use several scripting languages and platforms for automating tasks. This situation is undesirable, as I discuss in the following sections. To decide on a scripting language to use, you need to consider portability, understandability, power and flexibility, community support, and network applicability.

Portability

Portability means how many UNIX systems a scripting language can run on. Because most enterprise networks have several UNIX flavors (e.g., Linux, AIX, Solaris), your scripting language must support every platform that you support. Otherwise, you lose the most important advantage of using just one language: consistency.

When using shell scripts, select the Bourne shell (i.e., *sh*) or the Korn shell (i.e., *ksh*). The Bourne shell is ubiquitous; all UNIX systems offer the Bourne shell. Modern UNIX systems also offer the Korn shell, which has more advanced features than the Bourne shell.

Note: Don't use the C shell (i.e., *csh*). C shell scripts might behave differently across UNIX systems. In addition, C shell scripts are inherently insecure. For more information about C shell problems, see Bruce Barnett's essay, "Top Ten Reasons not to use the C shell" (<http://www.grymoire.com/Unix/CshTop10.txt>).

Understandability

The language you use needs to be *writable* and *readable*. *Writability* is how easy a language is to learn, whereas *readability* is how easily someone can understand what a script is supposed to accomplish. As you automate tasks you'll need to develop new scripts and fix existing scripts. Thus, you need to focus on writability to ease new development and readability to ensure that new staff understands previous script writers' intentions.

Creating readable scripts is important because staff turnover might result in the author of a crucial script or application being unavailable when the script later needs modification. To make scripts readable, you can use indentation, style consistency, and documentation. (And, you can later pull out the documentation to better document a process.)

Power and Flexibility

Another factor in selecting a language is power. Some administrators mistakenly believe that increased power means decreased writeability or readability. Scripting languages such as Python are powerful, flexible, and easy to use. You need to choose a language that doesn't unnecessarily restrict your ability to solve complex problems.

In addition, your scripting platform must support network operations. Even Bourne shell scripts can work over a network with tools such as Secure Shell (SSH), although the scripts are often messy. Some languages, such as Perl and Python, offer comprehensive network capabilities that ease the task of creating network-based management systems.

Community Support

When you select a scripting language, don't underestimate the importance of Internet community support. Popular scripting languages such as Perl and Python have huge online communities that can help your staff automate your systems. Although the community members can't provide hands-on attention, they can answer your questions about how to best approach problems.

Note: Internet support groups include USENIX and SAGE. Usenet newsgroups also provide useful information for new and experienced systems administrators and managers.

Some languages, such as Rexx, have a lot of support on various platforms. However, these languages don't have the in-depth support that other languages (e.g., Perl, Python, and Bourne) do.

Network Applicability

The scripting language that you prefer might not be the best language to support your network's systems. For example, Perl is a poor choice if most of your systems don't use or support Perl. Select the language that works best in your environment.

Note: Languages to consider include Perl, Python, PHP, and Bourne. You can find ample written or online documentation for all these languages. In addition, Perl and Python have large Internet user communities.

Focus on Security

One of the most common problems in automating systems management is not paying enough attention to security when building solutions. Administrators often sacrifice security because they're in a hurry to write and test a script. However, increasing automation doesn't necessarily mean reducing system and network security. In fact, automation should increase security. Security involves four areas: network security, environmental security, user security, and file security.

Network Security

Regardless of the scripting platform you use, you need to consider the effect of automating jobs over the network. In the past, administrators used rsh for remote system access for manual and automated tasks. Unfortunately, rsh has two security drawbacks: poor authentication control and the use of clear text for network communication. Despite these problems, some networks still use rsh for compatibility with older applications and scripts (and to maintain the status quo).

SSH offers a secure alternative to rsh that matches rsh's functionality. You can use SSH in an rsh-like fashion, so you don't need to change older scripts and applications. All scripts that perform administrative functions over a network, including scripts that work on the local network and scripts that operate over the Internet, should rely extensively or entirely on SSH.

Environmental Security

Most scripting languages let you invoke local UNIX tools (e.g., `/bin/cp` to copy a file). Administrators without scripting experience often rely on environmental defaults to run UNIX tools and other applications. However, you need to explicitly define the path to your commands or modify the scripts' `PATH`. (The `PATH` is where the script will search for commands.)

You can use a simple path statement to define a Bourne shell script's path:

```
PATH=/sbin:/usr/sbin:/bin:/usr/bin
```

You can also change environmental variables in other languages such as Perl.

Some literature suggests using explicit paths to system tools rather than relying on environmental paths. For example, instead of invoking the UNIX copy command (i.e., `cp`) as follows:

```
cp file1 file2
```

you can use:

```
/bin/cp file1 file2
```

However, using an explicit path can be difficult in a large, heterogeneous environment. Some tools, especially in BSD and System V (SysV), are located in different places.

A simple solution is to define the path. This approach relies on UNIX conventions rather than on knowing tools' locations. Most of the tools you need are in `/sbin`, `/usr/sbin`, `/bin`, or `/usr/bin`. You can also find administrator-installed tools in `/usr/local/sbin` and `/usr/local/bin`. In case a program is in `/usr/bin` rather than `/bin`, you can define an appropriate path as follows instead of using an explicit path:

```
PATH=/sbin:/usr/sbin:/bin:/usr/sbin
cp file1 file2
```

However, this solution doesn't encompass every situation. The two major UNIX branches (i.e., BSD and SysV) provide different interfaces and output for some of the same commands (e.g., `ps`). Thus, you often need to tailor your scripts for the tools you're using. You might need to write an abstraction layer between your scripted solution and the underlying UNIX tool. For more information about writing an abstraction layer, see the section "Use Abstraction."

Note: You need to protect IFS, especially in Set UID (SUID) scripts. (IFS is the variable that shells such as the Bourne shell use to determine how to expand an expression into a list of arguments.) If an attacker changes IFS before invoking your SUID script, he or she can cause incorrect or dangerous behavior. For example, if your script performs a `system()` call (e.g., in Perl) and an attacker sets IFS to `/`, your program call won't execute as expected and might invoke the wrong program with arguments that you didn't intend.

User Security

Most users aren't malicious. However, you shouldn't trust user input because users often make mistakes that create security risks.

The best defense against user error is to thoroughly check users' input on the command line, in environmental variables, and during program input. You can use two approaches to check user input: Allow everything that isn't explicitly disallowed, or allow nothing that isn't explicitly allowed (i.e., deny-by-default).

In Chapter 2 I discussed the importance of deny-by-default. This concept applies to all types of security. When you write scripts, you need to use deny-by-default to ensure valid user input. For example, if you're designing an application that requests a path to a file, you should allow only valid characters rather than disallow a set of characters that could be dangerous to your script's operation.

Some languages, such as Perl, provide explicit support for user input checking. In Perl, this process is called *taint checking*. With taint checking, you must check the inputted information before Perl lets you use the information. This powerful feature greatly increases your scripts' security if you use it properly.

File Security

Many scripts, especially advanced scripts, use various files for logging, scratch work, and data storage. You need to ensure these files' security to reduce or eliminate the possibility of an attacker using your scripts to compromise or damage your systems.

Umask

To ensure file security, you need to properly use the `umask` function. Umask is the default permissions you use to create files. To determine the permissions value, subtract the umask value from 777. For example, if the umask value is 022, you must use the permissions value 755 to create files. A permissions value of 755 means the owner can read, write, and execute, and others can only read and execute. This umask value ensures that other users can't write to newly created files. The best umask value to use in scripts is 077; this value ensures that only a file's owner has access to the file. Remember that scripts that run as root sometimes create trivial data that includes sensitive information that attackers can access.

Temporary and Data Files

Most administrative scripts create files (e.g., to temporarily store information from a series of system commands, to create a new file for the next script iteration). You need to restrict access to new temporary and long-term files to appropriate users.

Scripts use temporary files, also called scratch files, on a short-term basis. These files aren't common and typically don't last beyond the life of the scripts that generate the files. You need to ensure that your temporary files are valid and that only the generating script and necessary users can access the files.

Your first task is to properly create temporary files. Many scripts are designed to create a temporary file in /tmp, but a common mistake is using a non-unique filename. For example, a backup script might create a file named /tmp/backup-set, which might be a temporary filename that another running script is using. To solve this problem, most script writers use the running script's process identifier (PID) in the filename. For example, /tmp/backup-set becomes /tmp/backup-set.3433, where 3433 is the PID of the script creating the file.

Note: You can use various methods to find a PID. In Perl and Bourne shell scripting, you can use the special variable \$\$ to find a PID. Thus, you'd use /tmp/backup-set.\$\$ to create a file named /tmp/backup-set.pid, where pid is the PID of the script creating the file.

Unfortunately, using a PID doesn't guarantee that the filename is unique. For example, a file might be left from a previous invocation if a script fails ungracefully. Or, an attacker might create a set of temporary files that he or she knows will follow your program's guidelines in an attempt to compromise your script's execution. Be sure to check that the filename you want to use isn't already in use. You can use -f to check the filename in a Bourne shell script.

```
[ -f $tmpfile ] && touch $tmpfile
```

The following script defines a variable named \$tmpfile, which is a filename that includes the script's PID. The script then determines whether a file named \$tmpfile exists and creates the file if it doesn't exist.

```
PATH=/bin:/usr/bin umask 077
tmpfile=/tmp/script-name.$$

if [ -f $tmpfile ]; then
    touch $tmpfile
else
    echo "Can't create $tmpfile"
    exit 1
fi
```

Even with this solution, a race condition exists because an attacker can create /tmp/script-name.\$\$ between -f \$tmpfile and touch \$tmpfile. Although Bourne shell scripts have this security hole, more advanced scripting languages such as Perl avoid this problem because they have access to the open() system call. The open() system call creates a new file. You can use the open() system call in a mode in which the file to be created can't exist or the open() call fails.

Note: A system call is a UNIX programming function. System calls include open(), close()— which closes open files, and ioctl()—which provides an additional interface to file-system objects. Only the OS can make certain guarantees, such as an operation being atomic. An atomic operation occurs without interruption, so attackers don't have a chance to intervene during sensitive operations such as creating a file on a file system.

The best solution in Bourne shell scripts is to create a directory (or use `mktemp`, which I also discuss) in which to work, because `mkdir` will fail if the directory that `mkdir` attempts to create already exists. In my example, replace the temporary file with a scratch directory as follows:

```
PATH=bin:/usr/bin
umask 077

mkdir -m 700 /tmp/temp.$$ || exit 1

rm -rf /tmp/temp.$$
```

If `mkdir` fails (e.g., it can't create `/tmp/temp.$$` because the directory already exists), the script will fail and close. You can then safely create scratch files in `/tmp/temp.$$`.

Finally, you can use Todd Miller's `mktemp` tool (<http://www.mktemp.org>). `Mktemp` creates a temporary filename that's unique and is available only to the calling script.

Don't Reinvent the Wheel

Systems administration, especially UNIX administration, doesn't change much. Because UNIX is such an old OS, any problems you encounter probably already have documented solutions. You need to rely on previous UNIX systems administration knowledge.

You can easily find solutions in existing code for problems with popular scripting languages such as Perl and Python. Familiarize yourself with Internet community resources for your language (e.g., Comprehensive Perl Archive Network—CPAN—for Perl) so you know where to find solutions when problems arise.

Design Scripts for Failure

When you write scripts, especially scripts that modify system settings, design your scripts to fail gracefully. In programming, failing gracefully means that an application or script doesn't just fail when an error occurs but cleans up after itself and leaves the system in a consistent state. Imagine that you wrote a script to add new users, but instead of using a system tool the script directly modifies `/etc/passwd` and `/etc/shadow`. If the script modified `/etc/passwd` but didn't complete the changes to `/etc/shadow`, preexisting users might not be able to log on. You need to avoid situations in which a script failure makes the system unusable.

Succeed Quietly, Fail Loudly

You also need to consider how your administrators respond to failure. One of administrators' biggest problems is information overload. Administrators often have hundreds of reports to review each day, from various jobs running on their servers. Writing scripts that inform administrators of the scripts' successes might seem like a good idea, but administrators might start to ignore such frequent notices and therefore overlook a script failure.

Good script design means writing scripts that succeed quietly but fail loudly. That is, scripts shouldn't generate reports if the scripts run successfully. However, scripts that fail

should generate notices. If your scripts use cron to run, failure notification is simple because cron automatically emails scripts' output to the administrator.

Design your scripts to notify administrators of important system failures, but don't overwhelm administrators with excessive reports. Administrators pay more attention to notices if notices are the exception rather than the rule. Limiting notices to only script failures makes automation more effective.

Log from Scripts

You might need to log scripts' actions when they run. As I discussed previously, you should restrict automated notices to exceptions so that administrators don't become immune to log entries and start to ignore them. However, you must consider how automated jobs generate the necessary notices and log events. First, you need to differentiate the types of logs that automated jobs generate. In general, you can categorize log types as informational, warning, and error. Figure 1 shows the information flow to logs and administrators for each type of log.

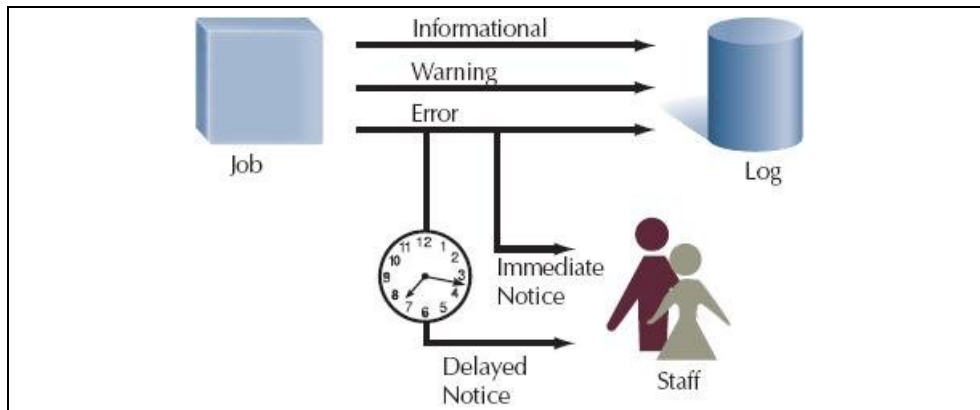


Figure 8-34. Information Flow to Logs and Administrators

Informational Logs

Informational notices notify administrators about scripts' actions. Administrators typically don't review informational logs on a regular basis. Rather than sending these logs directly to administrators, you should store the logs so that administrators can retrieve them in the event of an error.

The easiest place to store informational messages is in a log file, such as `/var/log/automated-job.log`. This solution works as long as the automated job doesn't run at the same time as other invocations of the same script. (That is, the solution works if only one script is running and logging to the message file.) If multiple jobs are logging to the same file, the scripts might interfere with one another when writing log messages. (In most UNIX systems, the OS guarantees that only small amounts of data can write atomically to a file.)

The best solution in most situations is to use syslog for automated job logging. Syslog prevents concurrency problems. In addition, using programs such as `rotatelog` (Linux) or

newsyslog (BSD) rotates syslog log files on a consistent basis and ensures that the log file system doesn't fill up.

You should also consider using a database, such as MySQL or PostgreSQL, to store log messages from automated jobs. These databases are free and reliable tools for systems administrators.

Warning Logs

A warning log generates when an automated job encounters an exception that isn't important enough to warrant an error message. For example, a warning notice might generate if a script detects that a disk still has space but is filling up. In this case the administrator needs notification but not immediately.

You can configure warning messages to email the administrator or log to a file for later review. If you log warning messages to a log file or database, you need to employ a log-monitoring tool (e.g., logcheck, logwatch) to generate alerts after the log-monitoring tool detects warning messages.

Error Logs

Error notices are urgent; administrators need to receive such notices immediately. An example error message is that the disk is full and the automated job is unable to complete.

Methods for sending immediate alerts include email and Microsoft Systems Management Server (SMS) paging. Keep in mind that when an error alert generates, the system might be under stress and therefore unable to send the alert message. To ensure that you receive error notification, you can use an outside monitoring server. With an external monitoring system, an automated job sends a completion notice. If the monitoring system doesn't receive the completion notice within an allotted time, the monitoring service generates an alert. This solution prevents the catch-22 that when automated jobs are failing, the system might also be failing.

Keep it Simple

UNIX is designed around a task-oriented toolset, in which each tool solves a specific problem. Your automated solutions should mirror this design. Rather than designing large, comprehensive systems to solve all your problems, use a task-focused script to address each problem. This approach increases your scripts' writeability and maintainability and helps administrators focus on tasks.

Another way to keep scripts simple is to restrict the user interface to the command line. This method reduces the amount of code necessary to handle user options and increases your solutions' usability because you can use other programs to invoke and control your scripts. Instead of placing the user interface directly into your task-oriented scripts, build menu systems around your scripts. This approach keeps each of your scripts, including the menu system, focused on a specific task that you can easily test.

Use Abstraction

Because you need to consider the variations in systems management and system interfaces in heterogeneous UNIX networks, use the appropriate levels of abstraction when designing scripts. The proper level of abstraction shouldn't affect a script's performance but should ensure that the script works properly across all your UNIX flavors, as Figure 2 shows. For example, if you manage SysV and BSD systems, you can write a `newdisk.sh` script that works on both platforms. As you add new UNIX flavors to your environment, you can modify the system-specific code under the abstraction layer to add support in your scripts.

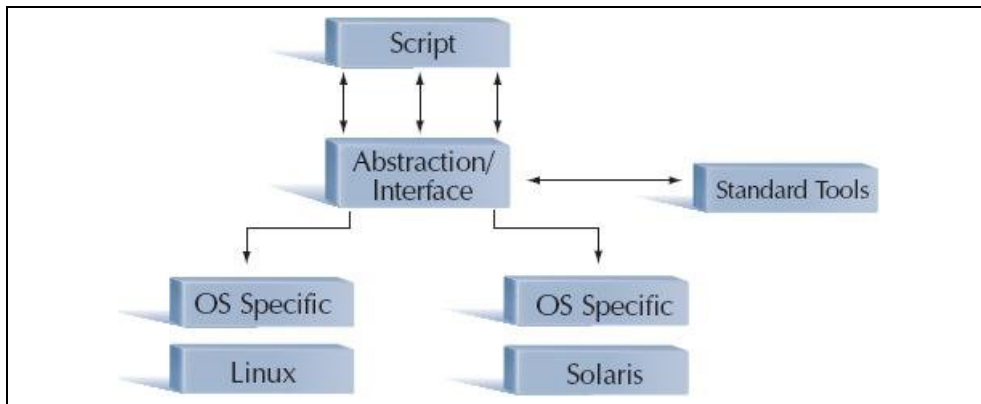


Figure 8-35. Abstraction Levels

For example, consider the difficulty in using `ps`. The `ps` command, which is available on all UNIX platforms, has various types of output depending on which UNIX flavor you're using. If your script must support `ps` across different UNIX flavors, you need to ensure that you can consistently access the output. Rather than parsing the output directly, a better solution is to write a function that parses `ps`'s output based on the system in use, as the following pseudo-Bourne shell code shows:

```
ps_capture() {
    capture=`ps`

    case $SYS do
        Linux)
            ...;
        Solaris)
            ...;
        SCO)
            ...;
    esac
}

ps_capture
```

Rather than calling `ps` directly, the script uses `ps_capture` to run `ps` and parse the output. The `ps_capture()` function then sets script variables to the necessary data. The script runs stably and consistently regardless of the underlying UNIX flavor.

Centralize Scripts

You need to maintain a location for accessing or publishing your scripts. Many organizations store scripts on an NFS server, as Figure 3 shows. Remote workstations and servers then mount the NFS file system and run the scripts directly from the mounted file system. This option works only if your servers can survive without the scripts if the NFS server fails.

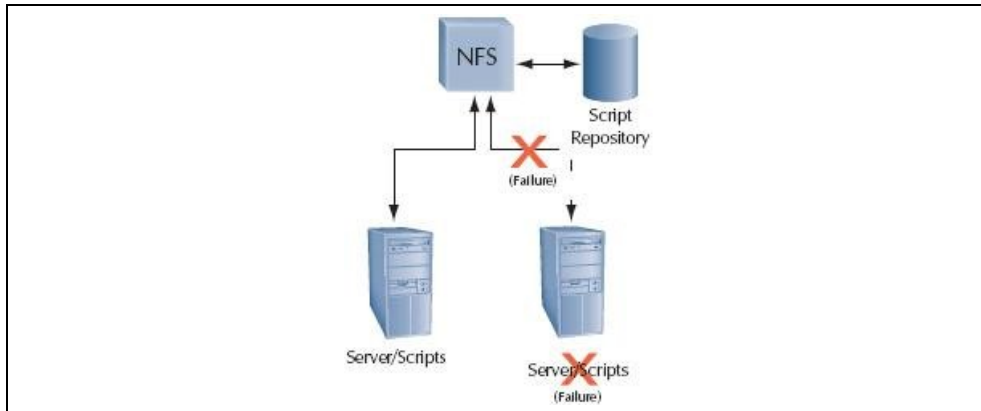


Figure 8-36. Storing Scripts on an NFS Server

Another effective solution for centralizing scripts is to maintain a central location for storage but distribute scripts across the network. Many administrators use rsync over SSH for this purpose. Rsync runs on a regular basis (e.g., twice a day) and pushes changes to the remote servers. If an NFS server fails, servers that rely on the NFS server won't operate correctly. But this problem doesn't occur because the remote servers continue to run even if the central storage location fails.

Conclusion

In this chapter I discussed how to best automate system and network management. In previous chapters I explained various best practices to help eliminate problems and increase your systems' efficiency. You can use these suggestions to turn even the most complex network into a self-managing system. For more information about systems administration and building secure and useful scripts, see *Essential System Administration*, *Practical UNIX & Internet Security*, and *UNIX Power Tools* (O'Reilly Media).